NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

BOOTSTRAP BASED SIGNAL DENOISING

by

Hasan Ertam Kan

September 2002

Thesis Advisor:	Monique P. Fargues
Co-Advisor:	Ralph D. Hippenstiel
Second Reader:	Roberto Cristi

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Bootstrap Based Signal Denoising	**5. FUNDING NUMBERS**
6. AUTHOR Hasan Ertam Kan	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

This work accomplishes signal denoising using the Bootstrap method when the additive noise is Gaussian. The noisy signal is separated into frequency bands using the Fourier or Wavelet transform. Each frequency band is tested for Gaussianity by evaluating the kurtosis. The Bootstrap method is used to increase the reliability of the kurtosis estimate. Noise effects are minimized using a hard or soft thresholding scheme on the frequency bands that were estimated to be Gaussian. The recovered signal is obtained by applying the appropriate inverse transform to the modified frequency bands. The denoising scheme is tested using three test signals. Results show that FFT-based denoising schemes perform better than WT-based denoising schemes on the stationary sinusoidal signals, whereas WT-based schemes outperform FFT-based schemes on chirp type signals. Results also show that hard thresholding never outperforms soft thresholding, at best its performance is similar to soft thresholding.

14. SUBJECT TERMS Denoising, Bootstrap, Kurtosis.	**15. NUMBER OF PAGES** 109
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

BOOTSTRAP BASED SIGNAL DENOISING

Hasan E. Kan
First Lieutenant, Turkish Army
B.S., Turkish Army Academy, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2002

Author:	Hasan E. Kan
Approved by:	Monique P. Fargues Thesis Advisor
	Ralph D. Hippenstiel Co-Advisor
	Roberto Cristi Second Reader
	John P. Powers Chairman, Electrical and Computer Engineering Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This work accomplishes signal denoising using the Bootstrap method when the additive noise is Gaussian. The noisy signal is separated into frequency bands using the Fourier or Wavelet transform. Each frequency band is tested for Gaussianity by evaluating the kurtosis. The Bootstrap method is used to increase the reliability of the kurtosis estimate. Noise effects are minimized using a hard or soft thresholding scheme on the frequency bands that were estimated to be Gaussian. The recovered signal is obtained by applying the appropriate inverse transform to the modified frequency bands. The denoising scheme is tested using three test signals. Results show that FFT-based denoising schemes perform better than WT-based denoising schemes on the stationary sinusoidal signals, whereas WT-based schemes outperform FFT-based schemes on chirp type signals. Results also show that hard thresholding never outperforms soft thresholding; at best its performance is similar to soft thresholding.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

ACKNOWLEDGMENTS

I wish to express my sincerest appreciation to my advisors Professor Monique Fargues and Professor Ralph Hippenstiel for their support and supervision.

I would like to thank Professor Roberto Cristi for his contributions.

Most importantly, I would like to thank my wife for her love, care and support. I would not have been able to do this without her.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The main goal of this thesis was to develop a denoising scheme to recover signals that are distorted by additive white Gaussian noise, regardless of the signal's sperctral content. Two approaches were investigated, a short-time Fourier transform and a wavelet transform-based approach. Both transforms are used to decompose a signal into several frequency bands and to minimize the noise in each band. Each frequency band content is tested for Gaussianity, which is accomplished by investigating the signal's kurtosis. It is well known that the signal kurtosis is equal to three only when the signal is Gaussian with zero mean. Therefore this parameter can help to provide a reliable determination whether or not a data sequence is Gaussian. However, the kurtosis estimate of a short data segment is not always reliable. The Bootstrap method is used in the kurtosis estimation to overcome this difficulty. The Bootstrap is a statistical scheme, which improves the reliability of a parameter estimate in situations where conventional techniques are not valid because of short data length issues. The Bootstrap uses data sequences that have the same length as the original signal, but are obtained by randomly resampling the data using replacement. These resampling is done many times and each data set is treated as repeated experiment. Then, the parameter of interest is estimated for each of these resampled sequences, to obtain a statistic for the parameter of interest. This resulting statistic can then be used to perform hypothesis tests on the parameter value.

If the band is estimated to be Gaussian (i.e., the noise is dominant), then thresholding is applied to the band to minimize the noise effects. Two thresholding techniques are considered, hard and soft threshold. Hard thresholding coefficients are 0 for Gaussian data and 1 for non-Gaussian data. Whereas soft thresholding coefficients are obtained by considering the data's closeness to Gaussianity. The closer to being Gaussian the band specific data is, the smaller the soft thresholding coefficients are and vice versa. The denoised signal is obtained by applying the appropriate inverse transformation.

Three different test signal types are selected to investigate the performances: a sinusoid, a chirp with constant amplitude, and a chirp with an RC time constant-like amplitude increase. The mean square error (MSE) and a distance measure defined between original and recovered signals are selected to compare performances. Results show FFT-based denoising schemes perform better than WT-based denoising schemes on the stationary sinusoid signal type, whereas WT-based schemes outperform FFT-based schemes on chirp type signals. Finally, results show that the soft thresholding scheme always performs at least as well as or better than the hard thresholding one.

I. INTRODUCTION

In many data transmission and storage systems, noise gets introduced into data, which reduces the signal quality. As a result, various filtering techniques have been designed to suppress noise contributions in order to improve the overall signal quality.

Fourier and Wavelet transforms decompose a noisy signal into several frequency bands. Traditional filter design methods have requirements on the frequency, magnitude and phase of the signal such as passband ripple, stopband attenuation, transition width, and phase constraints. The assumption behind these design criteria is that the signal is restricted to be in a certain frequency band and that the frequencies outside this specific band are treated as distortion. Note that this paradigm breaks down when signal and distortion terms overlap in frequency.

Denoising attempts to remove the noise and to recover the original signal regardless of the signal's frequency content. The basic idea is to look at each frequency band of interest and to minimize its noise effect by retaining the dominant component. The band is left untouched when the signal is dominant so as not to lose the signal component, while thresholding is applied when the noise is dominant.

This thesis discusses a denoising scheme that implements frequency band specific thresholding schemes using the Bootstrap method and the kurtosis. Chapter 2 presents the processing techniques and the Bootstrap method. Chapter 3 discusses the proposed denoising scheme. Simulation results are presented in Chapter 4. Finally, Chapter 5 presents conclusions.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. PROCESSING TECHNIQUES

Signal processing allows users to extract relevant information from a given signal. When the raw data does not allow extracting the desired information, transformations to another domain may be performed to do so. The most common transformation types are the Fourier and Wavelet transforms, which are discussed in the following two sections.

1. Fourier Analysis

Fourier analysis allows the representation of a given signal as a linear combination of complex sinusoids with different frequencies. For periodic signals this is called the *Fourier Series*. This representation becomes extended to the *Fourier Transform* when the signal is aperiodic. Both representations are discussed in the following subsections.

a Fourier Series

A periodic signal $x(t)$ with period T_0 may be represented as an infinite linear combination of complex exponentials:

$$x(t) = \sum_{k=-\infty}^{+\infty} a_k \, e^{jk\frac{2p}{T_0}t} \, , \tag{2.1}$$

where $f_0 = 1/T_0$ is the *fundamental frequency*, $e^{jk\frac{2p}{T_0}t}$ is called the k^{th} harmonic, and a_k is the k^{th} weight. This representation is called the *Fourier series* representation [1], where the set of coefficients a_k are called the *Fourier series coefficients* which are obtained by

$$a_k = \frac{1}{T_0} \int_{T_0} x(t) e^{-jk\frac{2p}{T_0}t} dt, \quad k = -\infty, \ldots, \infty . \tag{2.2}$$

The coefficients a_k's are a measure of the strength of the signal's components at the k^{th} harmonic of the fundamental frequency.

b. *Fourier Transform*

For aperiodic signals the Fourier series representation can be extended. Assuming that $x_p(t)$ is a periodic signal with a period T_0, and $x(t)$ represents one period of $x_p(t)$, then as T_0 increases $x_p(t)$ is identical to $x(t)$ over a longer interval. Therefore, in the limiting case [2]

$$x(t) = \lim_{T_p \to \infty} x_p(t) . \tag{2.3}$$

Replacing the limits of the integral in equation (2.2) by $-\infty$ and ∞, and multiplying both sides by T_p, leads to the *Fourier Transform* of *x(t)* given by

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2p\,ft} dt , \tag{2.4}$$

where $f = k / T_p$ and $X(f) = T_p a_k$. The *Inverse Fourier Transform* is used to recover the original signal $x(t)$ from $X(f)$, and is defined as

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2p\,ft} df . \tag{2.5}$$

The *Discrete Time Fourier Transform* (DTFT) is defined for discrete-time signals as

$$X(w) = \sum_{n=\infty}^{\infty} x(n)\, e^{-jwn} , \tag{2.6}$$

and the corresponding inverse is given by

$$x(n) = \frac{1}{2p} \int_{2p} X(w)\, e^{jwn} dw . \tag{2.7}$$

It should be noted that the DTFT is periodic with period $2p$.

The *Discrete Fourier Transform* (DFT) for a finite time discrete-time signal *x(n)* with *n=0,…,* $N-1$ is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2p\,kn/N} . \tag{2.8}$$

Its inverse is given by

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, e^{j2p\,kn/N}. \qquad (2.9)$$

The *Fast Fourier Transform* is a computationally efficient implementation of the Discrete Fourier Transform when the signal length N is a power of two. The Discrete Fourier Transform requires N^2 multiplications whereas the Fast Fourier Transform requires $(N/2)\log_2 N$ multiplications.

c. Short-Time Fourier Transform

The basic Fourier Transform allows the passage from the time domain to the frequency domain. However, one may need to preserve both the time and the frequency information contained in non-stationary signals. Unfortunately, the basic Fourier Transform does not provide such dual information.

Time localization can be introduced by windowing the signal before using the Fourier Transform, where the window size is selected short enough to assume the signal inside the window is stationary. Then, the Fourier transform may be implemented on each windowed signal portion. The resulting transformation is called the *Short-Time Fourier Transform* and expressed as

$$S(t, f) = \int_t x(t) w(t - t)\, e^{-j2p\, ft} dt, \qquad (2.10)$$

where $w(t - t)$ denotes the sliding window centered around t. Note that $S(t, f)$ provides a two-dimensional representation of the signal frequency information at various times t and is a “local” spectrum of the signal $x(t)$ around the analysis point t [3]. Many different window types may be selected, depending on the characteristics of the Short-Time Fourier Transform desired. In addition, the window length determines the resolution in time and in frequency; good time resolution requires a short window, whereas good frequency resolution requires a long window. The joint time-frequency resolution of the Short-Time Fourier Transform is limited by the uncertainty principle. A short time window results in a loss of frequency resolution, and vice versa.

2. Wavelet Analysis

A *wavelet* is defined as a small wave which has its energy concentrated in time and frequency [4]. Such a characteristic is useful for the analysis of transient, non-stationary or time-varying phenomena. By comparison, sinusoidal functions used in Fourier Analysis have a constant amplitude.

The Wavelet Transform (WT) provides an alternative to the Short-Time Fourier Transform (STFT), as it uses short windows at high frequencies and long windows at low frequencies [5], while the STFT uses windows of constant size, as illustrated in Figure 1. Note that the Wavelet Transform still satisfies the uncertainty principle, however, the time resolution becomes arbitrarily good at high frequencies, while the frequency resolution becomes arbitrarily good at low frequencies. The following subsections discuss different types of Wavelet Transform that are used for continuous or discrete time signals.

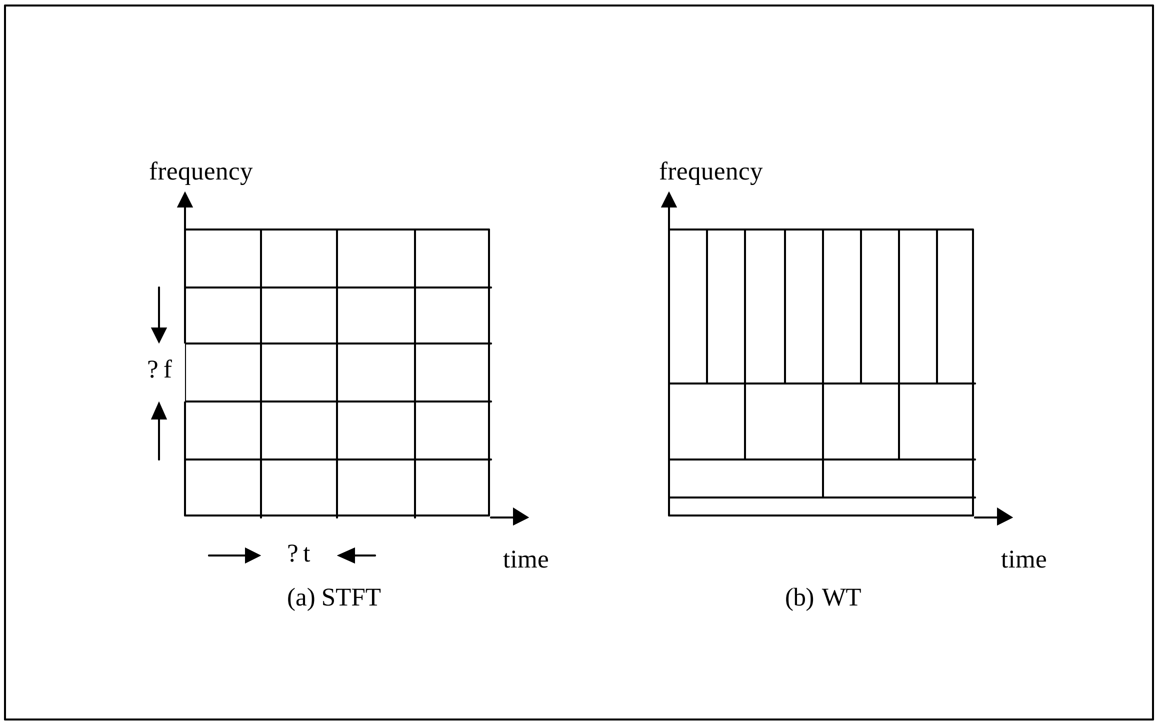

Figure 1. (a) Time-Frequency resolution for the STFT, (b) Time-Frequency resolution for the WT.

a. The Continuous Time Wavelet Transform

The *Continuous Time Wavelet Transform* may be applied to continuous signals and two dimensions in the transfer domain are continuous. It is similar to the Fourier Transform in that, it is obtained by projecting the signal onto a basis function. However, the Continuous Time Wavelet Transform projects the signal onto scaled and shifted versions of the wavelet function while the Fourier Transform uses complex exponentials as basis functions. The Continuous Time Wavelet Transform is defined as

$$C(s,t) = \frac{1}{\sqrt{s}} \int_{-\infty}^{\infty} x(t) y\left(\frac{t-t}{s}\right) dt , \tag{2.11}$$

where $y(t)$ is the wavelet function, t denotes the shift in time and s is the scale factor that denotes compression or expansion in time. The inverse transform for finite K is obtained by

$$x(t) = K \int\int \frac{1}{s^2} C(s,t) y\left(\frac{t-t}{s}\right) ds\, dt , \tag{2.12}$$

where the parameter K is given by

$$K = \int \frac{|\Psi(w)|^2}{|w|} dw , \tag{2.13}$$

and $\Psi(w)$ is the Fourier Transform of the wavelet function $y(t)$. [6]

b. The Discrete Time Wavelet Transform

The Discrete Time Wavelet Transform is the discrete time version of the Continuous Wavelet Transform. It is used for discrete time signals and the dimensions of the transform domain are discrete as well. The Discrete Wavelet Transform is obtained as

$$C(a,b) = \sum_n \frac{1}{\sqrt{a}} x(n) y\left(\frac{n-b}{a}\right) , \tag{2.14}$$

where a, b and n are the discrete parameter versions of s, t and t given in Equation (2.11), respectively. The scaling factor a has another restriction as $a = a_0^j$ where

$j=0,1,\ldots,\log_2$(*signal length*). The common choice for a_0 is 2, as it allows for fast algorithms.

c. Mallat's Algorithm

The Discrete Wavelet Transform, can be implemented by using Mallat's algorithm when $a_0 = 2$ [4], and is illustrated in Figure 2 for a three-level decomposition. The signal is passed through a high-pass and a low-pass filter, both of which have a bandwidth of half the signal spectrum. Then, following the Nyquist's rule, the outputs of the filters are subsampled by two. The subsampled high-pass filter output is called the *Detail Sequence* and the subsampled low-pass filter output is called the *Approximation Sequence*. This procedure may be recursively applied to the approximation sequence obtained at previous levels. Note that a signal of length 2^j can be decomposed only j times, because the approximation sequence has only one sample left after j levels.

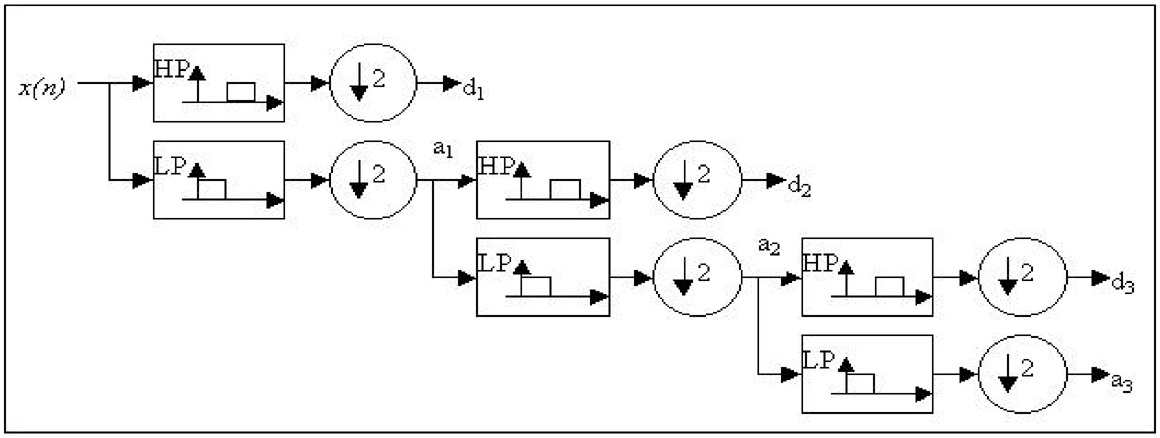

Figure 2. Mallat's algorithm.

B. HIGHER ORDER STATISTICS

1. Moments and Cumulants

The first four moments for a real valued and stationary signal $x(n)$ are given by

$$m_1 = E\{x(n)\}, \tag{2.15}$$

$$m_2(t_1) = E\{x(n)x(n+t_1)\}, \tag{2.16}$$

$$m_3(t_1,t_2) = E\{x(n)x(n+t_1)x(n+t_2)\}, \tag{2.17}$$

$$m_4(t_1,t_2,t_3) = E\{x(n)x(n+t_1)x(n+t_2)x(n+t_3)\}. \tag{2.18}$$

The first two moments are equal to the mean and correlation functions respectively. The first four cumulants are given by

$$c_1 = m_1, \tag{2.19}$$

$$c_2(t_1) = m_2(t_1) - m_1(t_1)^2, \tag{2.20}$$

$$c_3(t_1,t_2) = m_3(t_1,t_2) - m_1[m_2(t_1) + m_2(t_2) + m_2(t_2 - t_1)] + 2(m_1)^3, \tag{2.21}$$

$$\begin{aligned} c_4(t_1,t_2,t_3) = m_4(t_1,t_2,t_3) &- m_2(t_1)m_2(t_3-t_2) - m_2(t_2)m_2(t_3-t_1) \\ &- m_2(t_3)m_2(t_2-t_1) - m_1[m_3(t_2-t_1,t_3-t_1) \\ &+ m_3(t_1,t_2) + m_3(t_1,t_3) + m_3(t_2,t_3)] \\ &+ (m_1)^2[m_2(t_1) + m_2(t_2) + m_2(t_3) + m_2(t_3-t_1) \\ &+ m_2(t_3-t_2) + m_2(t_2-t_1)] - 6(m_1)^4. \end{aligned} \tag{2.22}$$

Note that the second and third order cumulants are identical to the second and third order moments respectively when $x(n)$ is a zero-mean process.

Cumulants have properties that make them more desirable than moments. Some of these properties are

a. Each cumulant is independent of all lower order cumulants.

b. All cumulants of order greater than two are equal to zero for Gaussian processes. Hence, any Gaussian process is completely characterized by its first two cumulants. Therefore higher-order cumulants can be used to estimate the degree of non-Gaussianity of a process.

c. Cumulants of the sum of two independent statistical processes are equal to the sum of their respective cumulants. [6]

2. Variance, Skewness and Kurtosis Measures

Setting t_1, t_2, t_3 equal to 0 in (2.20), (2.21), (2.22), and assuming that m_1=0, leads to the variance g_1, skewness g_2, and kurtosis g_3 measures:

$$g_2 = E\left\{x(n)^2\right\} = c_2(0), \tag{2.23}$$

$$g_3 = \frac{E\left\{x(n)^3\right\}}{s^3}, \tag{2.24}$$

$$g_4 = \frac{E\left\{x(n)^4\right\}}{s^4}, \tag{2.25}$$

where s is the standard deviation around the mean of the signal. [7]

C. THE BOOTSTRAP

The Bootstrap is a powerful technique for assessing the accuracy of a parameter estimator in situations where conventional techniques are not applicable [8]. In many applications, one needs to estimate one or more parameters of a random process, and/or calculate some statistical parameters such as the mean or variance. Most of the estimation techniques used for this purpose assume that the set of samples used in the estimation is large enough to reach asymptotic results. However, in practice this assumption usually does not hold as the sample set may not be large enough or the samples may be non-stationary. The Bootstrap scheme randomly reassigns the observations, recomputes the estimates many times, and treats these reassignments as repeated experiments. In this section, first the basic Bootstrap principle is stated. Next, the usage of Bootstrap in estimating the confidence interval for a parameter is discussed, and finally this discussion is extended to hypothesis testing.

1. Basic Principle

Let $x=\{x_1, x_2,\ldots, x_n\}$ be a collection of *n independent* and *identically distributed* random variables drawn from an unknown distribution, *D*. Let *p* denote an unknown statistical parameter of *D* such as the mean or the variance, and $\hat{p}$ denote an estimator of *p*, calculated from *x*. If the estimate $\hat{p}$ is to be used in place of the real parameter *p*, it may be important to know the sampling distribution of $\hat{p}$. The distribution of $\hat{p}$ may be estimated with the Bootstrap method, and is obtained by resampling many times from a distribution $\hat{D}$, chosen to be close to *D*, such that $\hat{D}$ approaches *D* as $n \to \infty$. Note that the choice of $\widehat{D}$ is not unique. If the type of *D* is known but its statistical parameters are not known, then $\widehat{D}$ is chosen as a distribution of the same type as *D*, with the statistical parameters obtained from x. For example, if we know that the data is Gaussian but do not know its mean and variance, we perform the resamplings assuming the data to have the same mean and variance as x has. This approach is called the *parametric Bootstrap*. If nothing is known about *D*, the resamplings are drawn from x with replacement, so that each value in a resample set has probability equal to *1/n*. This approach is called the *nonparametric Bootstrap*.

2. Parameter Confidence Interval

The Bootstrap principle may also be applied to obtain a $(1-a)100\%$ confidence interval for the parameter *p*. First, the data set is resampled many times such that each resampled set is of the same size as the data. Next, an estimator $\widehat{p}_k$ is obtained from each resampled set, where $k = 1,..., N$ and *N* is the number of repetitions. Then, the estimates $\widehat{p}_k$ are sorted in increasing order. Finally the indices of the lower limit $\widehat{p}_L$ and the upper limit $\widehat{p}_U$ of the estimates in

$$P(\widehat{p}_L \leq p \leq \widehat{p}_U) = 1-a \tag{2.26}$$

are obtained by using

$$L = \left\lfloor \frac{Na}{2} \right\rfloor \tag{2.27}$$

and

$$U = N - L + 1, \tag{2.28}$$

where $\lfloor A \rfloor$ denotes the integer part of the value *A*.[8]

3. Hypothesis Testing

a. Description

Suppose one needs to perform a hypothesis test, such as $H : p \leq p_0$ against the hypothesis $K : p > p_0$, where p_0 is given. A new statistic, defined as

$$\hat{T} = \frac{\hat{p} - p_0}{\hat{s}} \tag{2.29}$$

may be selected, where $\hat{s}$ is the estimator of the standard deviation s of $\hat{p}$. The estimator for the standard deviation $\hat{s}$ of $\hat{p}$ will be defined later.

To perform the hypothesis testing, one first draws random sequences $x_1^*, x_2^*, ..., x_N^*$, of the same size as x, with replacement from x. Note that * does not denote complex conjugation, but means that this is a resampled set, or a parameter obtained from a resampled set. Then the statistic $\hat{T}^*$ is estimated from each sequence x^* as

$$\hat{T}^* = \frac{\hat{p}^* - \hat{p}}{\hat{s}^*}, \tag{2.30}$$

where $\hat{p}^*$ and $\hat{s}^*$ are estimated parameters obtained from the resample x^*, instead of x, and the constant p_0 is replaced with $\hat{p}$. Note that $\hat{s}$ is included as a scale factor in the calculation of $\hat{T}$. Dividing by $\hat{s}$ is called *Bootstrap pivoting* and it is done to ensure $\hat{T}$ is asymptotically pivotal when $n \to \infty$, i.e., the asymptotic distribution of $\hat{T}$ does not depend on any unknown parameters. Replacing p_0 with $\hat{p}$ and using Bootstrap pivoting is important because the Bootstrap distribution of $\hat{T}^* = (\hat{p}^* - \hat{p}) / \hat{s}^*$ is a better approximation to the distribution of $\hat{T} = (\hat{p} - p_0) / \hat{s}$ under H, than the Bootstrap distribution of $S^* = \hat{p}^* - \hat{p}$ is to the distribution of $S = \hat{p} - p_0$ under H [8, 9]. Next, the set of test statistics $\hat{T}_1^*, \hat{T}_2^*, \ldots, \hat{T}_N^*$ are sorted by increasing order, and the hypothesis H is rejected if $\hat{T} > \hat{T}_{(M)}^*$, where M is chosen according to N and the level of significance a as [8, pp. 62]

$$M = (N + 1)(1 - a). \tag{2.31}$$

Note that the test statistic $\widehat{T}$ is given by

$$\widehat{T} = \frac{\left|\widehat{p} - p_0\right|}{\widehat{s}}, \tag{2.32}$$

when the hypotheses to be tested are $H : p = p_0$ against $K : p \neq p_0$.

b. Estimation of the Standard Deviation for $\widehat{p}$

The parameter $\widehat{s}$ can be estimated by using the Bootstrap. Towards that end, resamples of the same length as the data set x, called x^* are drawn from x randomly with replacement, to obtain a total of B resamples. After resampling, the Bootstrap estimates $\widehat{p}^*$ are calculated in the same manner as $\widehat{p}$ was, but with resamples x^* instead of x. As a result, the standard deviation $\widehat{s}$ of $\widehat{p}$ is estimated by

$$\widehat{s} = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\widehat{p}_b^* - \frac{1}{B}\sum_{b=1}^{B}\widehat{p}_b^*)^2}. \tag{2.33}$$

In this chapter we presented the processing techniques that we used in the proposed denoising scheme. In the next chapter we will discuss the proposed denoising scheme.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DENOISING USING THE KURTOSIS AND THE BOOTSTRAP

A. INTRODUCTION

Fourier and Wavelet transforms decompose a signal into several frequency bands. As a result, white Gaussian noise components affect all frequency bands. The proposed technique examines each frequency band and tends to minimize the white noise contribution. First, we obtain the signal time–frequency representation (or time-scale representation in the Wavelet transform case). Next, each frequency band is tested for Gaussianity, and thresholding is performed on the spectral location found to be Gaussian. Last, the signal is transformed to the time domain using the appropriate inverse transform. The following sections discuss two different implementations using Fourier and Wavelet transformations.

B. FAST FOURIER TRANSFORM BASED DENOISING

The FFT-based denoising scheme is performed in four steps, as illustrated in Figure 3. For computational convenience, the signal length is assumed to be 512 points. A longer data set could have been used, resulting in more frequency bands, and more computations. A data set of minimum length of 512 allows separation in time and frequency. A shorter segment will not provide a reasonable number of time-frequency cells.

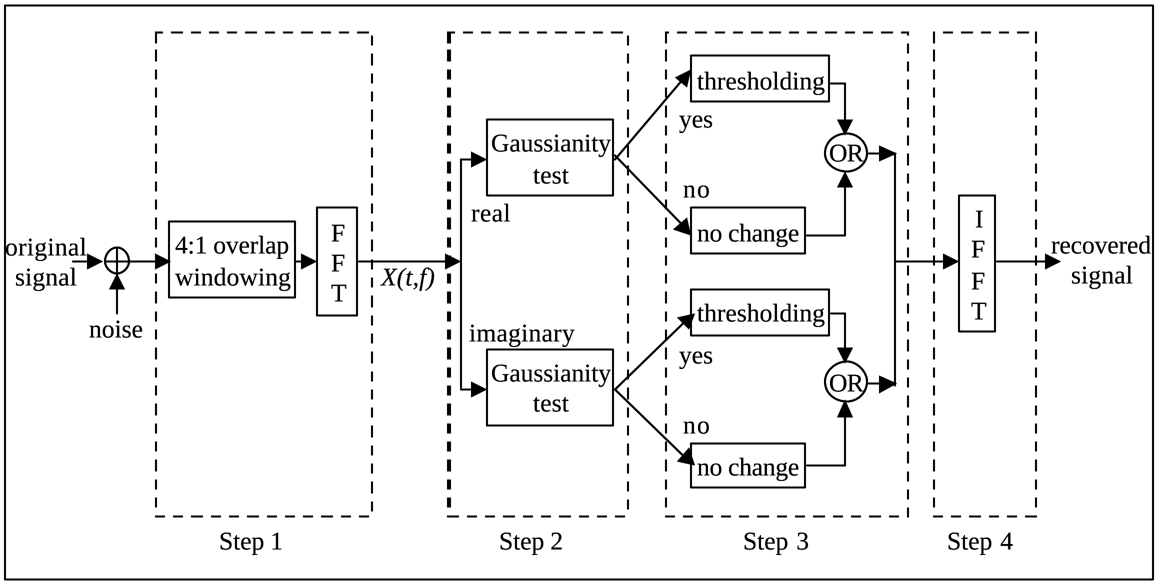

Figure 3. FFT-based denoising scheme.

1. Short-Time Fourier Transform Step

In the first step, the data is divided into 32 data-point segments and weighted by a triangular window. A four-to-one overlap is used to obtain a reasonable data size. Then, each segment is transformed to the frequency domain with the Fast Fourier Transform. This results in 17 frequency bands. A larger data set would have allowed for a larger segment size. The STFT information is contained in a matrix of dimensions 61 by 32 elements for a data length of 512 points, where the first number corresponds to the time dimension and the second one corresponds to the frequency dimension.

2. Gaussianity Test Step

The second step tests the Gaussianity of the transformed values in each frequency band, where real and imaginary parts of the data are tested separately with the kurtosis. Recall that the normalized kurtosis value for a Gaussian data sequence is equal to 3. Hence, looking at the kurtosis value may give an idea of a sequence's Gaussianity, provided the data length is sufficient for the estimation to be meaningful. However, the sequences in this particular scheme are of length equal to 61, which may not be sufficient for meaningful estimations. Therefore, the Gaussianity test is implemented as a hypothesis test, with the hypothesis $H: kurtosis of the data = 3$, and the hypothesis $K: kurtosis of the data \neq 3$. The sequence is found to be Gaussian when the hypothesis H is true, meaning the kurtosis value for the data sequence is 3 within the specified confidence interval. However, the sequence is found to be non-Gaussian when hypothesis K is true, meaning the kurtosis is not equal to 3 within the selected confidence interval. This hypothesis testing is performed using the Bootstrap method discussed earlier in Chapter 2. An empirical confidence interval $a = 0.05$ was selected here.

3. Thresholding Step

Signal components are found in a particular sequence when some frequency bands are estimated to be non-Gaussian for both real and imaginary parts. Thresholding is applied to each band estimated to be Gaussian, to minimize noise effects in these bands. Two thresholding schemes are considered below.

a. *Hard Thresholding*

Hard thresholding zeroes out all values in the frequency band that is found to be Gaussian. The hard thresholding coefficient is

$$c_h = \begin{cases} 0, & \text{if thebandisGaussian} \\ 1, & \text{if thebandisnotGaussian.} \end{cases} \tag{3.1}$$

b. *Soft Thresholding*

Soft Thresholding is obtained by multiplying values in the specific frequency band that is found to be Gaussian, by a coefficient between 0 and 1. Using a coefficient of 0 is the same as hard thresholding, whereas using a coefficient of 1 is the same as leaving the frequency band undisturbed. The soft thresholding coefficient is calculated using

$$c_s = \frac{\left|3 - \widehat{g_4}\right|}{1.5}, \tag{3.2}$$

where $\widehat{g_4}$ is the bootstrapped kurtosis of the particular frequency band, and $|\ |$ denotes the absolute value. The bootstrapped kurtosis value $\widehat{g_4}$ is limited not to exceed 4.5, which will be explained later. The bootstrapped kurtosis value $\widehat{g_4}$ is obtained by using the Bootstrap principle, which calls for resamplings from the data set many times with replacement, to obtain N resampled sets of the same length as the original data set. Next, the kurtosis value for each resample is found. Finally, the Bootstrapped kurtosis $\widehat{g_4}$ is defined as the estimated mean obtained from N kurtosis values. It should be noted that the thresholding coefficient c is a function of the frequency band's degree of Gaussianity. Equation (3.1) shows that the coefficient c gets closer to 0 as the bootstrapped kurtosis value $\widehat{g_4}$ for a specific frequency band gets closer to the theoretical value 3, and vice versa. Therefore, the closer a frequency band gets to being Gaussian, the smaller contribution it has after soft thresholding.

Recall that frequency bands to be thresholded are those that passed the Gaussianity test. Therefore, one would expect their corresponding bootstrapped kurtosis estimates to be close to the theoretical value 3. However in some cases, the estimated bootstrapped kurtosis value may be far off from 3. For example, a few values in the band may be higher than the others especially when a frequency bin contains a short data segment. In this case, the band may pass the Gaussianity test as most of the band is Gaussian, but may still have a bootstrapped kurtosis value high enough to obtain a thresholding coefficient c greater than 1. Note that a thresholding coefficient greater than 1 would amplify noise contributions in the frequency band, instead of suppressing them. Therefore, the bootstrapped kurtosis value $\widehat{g}_4$ is limited to 4.5 in (3.1) to avoid such potential noise amplification. This value is chosen empirically. Thus, if the bootstrapped kurtosis value of a specific frequency band is greater than 4.5, the data point with the largest absolute value in that band is stored away and replaced with a zero. Then, the bootstrapped kurtosis for that band is estimated again. This procedure is repeated until a kurtosis value less than or equal to 4.5 is obtained. However, the number of repetitions should be limited to ensure that most of the data is not zeroed out, to prevent bootstrapped kurtosis estimation problems. An empirical limit of one third of the number of time frames defined in the STFT. Note that if the estimated kurtosis value is still greater than 4.5 after that many iterations, then it is set to 4.5, which provides a thresholding coefficient equal to 1. This causes no change on the frequency band under consideration.

After multiplying the frequency band with the thresholding coefficient, the values that were removed to limit the kurtosis estimation are reinserted into their original locations. This allows for the non-Gaussian values, which may correspond to the signal components, to not be affected by the thresholding step.

4. Inverse Fourier Transform

Real and imaginary parts of the frequency bands are combined to form the time-frequency representation matrix after the thresholding step, and the recovered signal is obtained using the inverse Fourier transform.

C. WAVELET TRANSFORM BASED DENOISING

The Wavelet transform-based denoising scheme is similar to the FFT-based scheme, as illustrated in Figure 4. Note that only real valued transforms are obtained in this case due to the real wavelet transform kernel form selected in our work (Daubechies wavelets of order 3, 4 or 5). The choice of Daubechies wavelets was motivated by the earlier works in GSM signal denoising reported by Aktas and Mantis [10, 11]. This denoising scheme is performed in four steps.

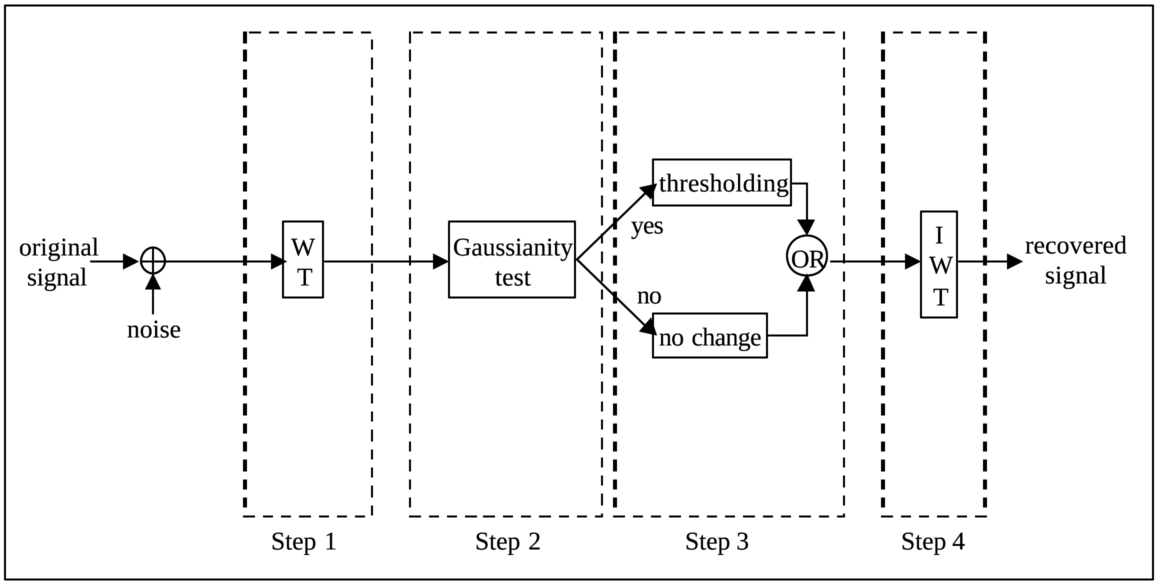

Figure 4. WT-based denoising scheme.

1. The Wavelet Transform

First, the Wavelet transform is applied to the noisy signal. Three-, four-, or five-level decompositions are considered in this work, because simulations showed that higher-level decompositions do not improve performance, given the data length considered (512).

2. Gaussianity Test

The approximation and detail coefficients of a Gaussian data set remain Gaussian [12]. Thus, detail and approximation coefficients are tested for Gaussianity and a decision is made for each.

3. Thresholding

The same thresholding procedure as that was considered earlier for the FFT-based denoising scheme is applied to the detail coefficients. However, thresholding the approximation coefficients may result in removing a significant portion of the signal. One can use one of the following four schemes to threshold the approximation coefficients to minimize potential distortion: Apply the thresholding scheme as that used on the detail coefficients, leave coefficients unperturbed, filter the approximation coefficients using a median filter of order three, or use a predictor of order two. The last two options were investigated as ways to smooth the approximation coefficients but they did not perform as good as the first two. Results for these operations are discussed in the next chapter.

4. Inverse Wavelet Transform

Finally, updated approximation and detail coefficients are inverse Wavelet transformed to obtain the recovered signal.

In this chapter the proposed denoising scheme was discussed in detail. This scheme was tested using three test signals. The test signal descriptions and the simulation results are presented in the next chapter.

IV. SIGNAL DESCRIPTION AND SIMULATION RESULTS

The denoising scheme presented in the previous chapter was tested using three test signals. The codes used for the simulations are presented in Appendix A. In this chapter the test signals are described and the simulation results are summarized.

A. SIGNAL DESCRIPTION

Three different types of test signals are used: a sinusoid, a chirp with constant amplitude, and a chirp with amplitude increasing in an RC time constant fashion, not to exceed a given maximum value. These signals are described below.

1. Sinusoidal Signal

The frequency of the sinusoidal signal is selected randomly for every trial, where the frequency range is limited to avoid aliasing and DC signals. Figure 5 shows an example where $f_0 = 10/512$, and the sampling frequency $f_s = 1$.

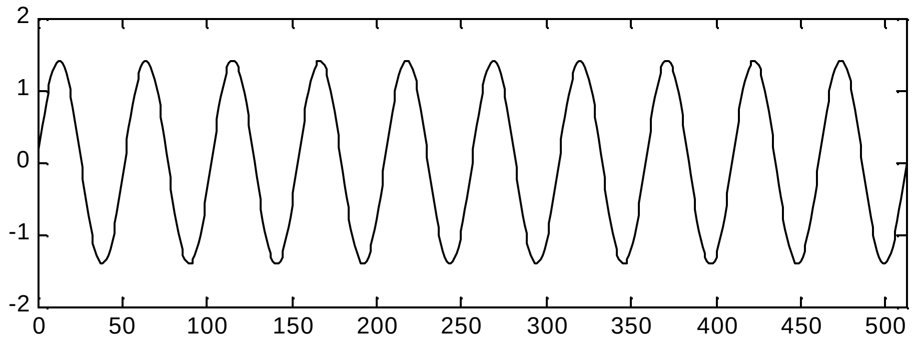

Figure 5. Sinusoidal test signal; frequency $f_0 = 10/512$, sampling frequency $f_s = 1$.

2. Constant Amplitude Chirp

The constant amplitude chirp used in the simulations is obtained by

$$s(t) = \sin\left(\frac{3470}{40+t}\right). \tag{4.1}$$

The signal sampled with sampling frequency $f_s = 1$ is shown in Figure 6.a.

3. Chirp With an RC Time Constant-Like Amplitude Increase

This signal has a frequency starting with a high value and decreasing in time. However, the amplitude of the signal starts with a small value and increases with time. The increasing amplitude chirp is obtained by

$$s(t) = \sqrt{\frac{(t+9)(512-t)}{521}}\sin\left(\frac{2p\,547.05}{t+35.05}\right). \tag{4.2}$$

This test signal sampled with sampling frequency, $f_s = 1$, was obtained from Wavelab [13] and is illustrated in Figure 6.b.

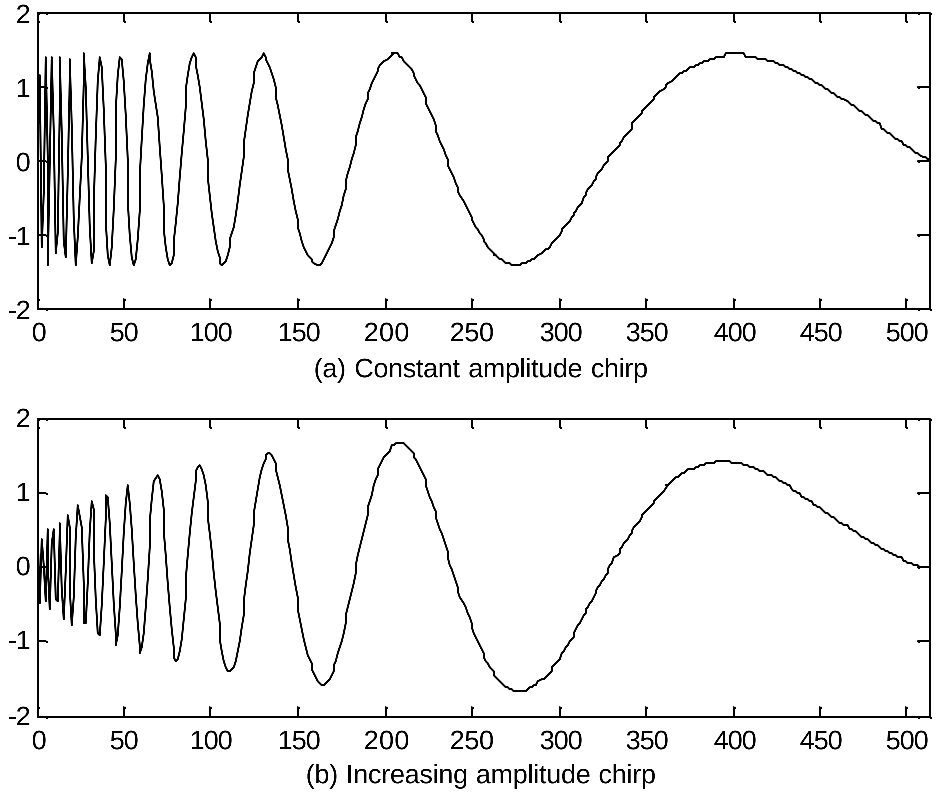

Figure 6. (a) Second test signal, (b) Third test signal.

B. SIMULATION RESULTS

MATLAB [14] simulations were performed to test the performance of the proposed schemes. One hundred trials are considered for both schemes for signal-to-noise ratio (SNR) values ranging between –6 dB and 6 dB. To perform the hypothesis test in the second step of both FFT-based and WT-based schemes, the Bootstrap MATLAB Toolbox [15] was used. The MATLAB code is included in Appendix A.

The mean square error (MSE) and cross-correlation coefficients between original and recovered signals were selected as performance criteria. The MSE was initially selected as it is commonly used in signal processing applications to measure signal differences in the time domain. It is defined as:

$$MSE = \frac{1}{M}\sum_{j=1}^{M}\left(\frac{1}{N}\sum_{i=1}^{N}\left|s_j(i) - \widehat{s}_j(i)\right|^2\right), \tag{4.3}$$

where M is the number of trials, N is the signal length, $s_j(i)$ and $\widehat{s}_j(i)$ are the i^{th} data sample of the original signal and the recovered signal at j^{th} trial, respectively. Note that the MSE may not always be useful in evaluating actual performances. For example, in a denoising scheme when the recovered signal is close to the original version in most of the signal duration but it is very different for a short duration, the overall performance of the scheme can be satisfactory but at the same time the MSE may be large. In addition, simulations showed the MSE performances to be very similar on a large portion of the schemes investigated. Therefore, we considered an additional performance criterion, based on the cross-correlation coefficient to complement the information given by the MSE criterion. Recall that the normalized cross-correlation coefficient is commonly used to evaluate signal similarities and is defined as:

$$r = \frac{1}{\sqrt{\sum_{i=1}^{N}|s(i)|^2 \sum_{i=1}^{N}|\hat{s}(i)|^2}}\sum_{i=1}^{N} s(i)^* \hat{s}(i), \tag{4.4}$$

where $()^*$ denotes complex conjugation. Note that the cross-correlation coefficient r should be equal to 1 for a perfectly reconstructed noise-free signal. The closer the magnitude of r gets to 0, the worse the denoised signal will be. The distance measure between 1 and the normalized cross-correlation coefficient is given by

$$d = \frac{1}{M}\sum_{j=1}^{M}[1-r_j]^2, \tag{4.5}$$

where M is the number of trials and r_j is the cross-correlation at lag zero for the j^{th} trial. The resulting measure d is the additional criterion used to evaluate the denoising schemes' performances in extracting the noise-free signal from the noisy signal.

In our simulations sometimes the MSE performances of various schemes were close to each other and it was difficult to make a distinction among them. In this kind of cases the distance measure r was used in making a distinction.

The proposed denoising schemes are compared with the original Wavelet shrinkage algorithm, which was introduced by Donoho and Johnstone [16, 17] to denoise signals embedded in additive white Gaussian noise with unit variance. Note that our schemes do not require knowledge of the noise variance levels to be applied, which is not the case for the original Donoho and Johnstone scheme. Also note that variable noise variances had been used in the simulations for the proposed scheme. Therefore the setup for the simulations needed to be changed before comparing the simulations for the proposed schemes with the Wavelet shrinkage algorithm. The Wavelet shrinkage algorithm is defined and comparison results are presented in Appendix C.

1. Fast Fourier Transform-Based Denoising

Performance criteria for the three test signals are illustrated for both thresholding options, in Figures 7 to 9. Examples of noisy signals and their recovered versions obtained by FT-based denoising scheme are included in Appendix B.

a. Sinusoidal Signal

Results, shown in Figure 7, indicate that hard and soft thresholding schemes have similar performances at all SNR values considered.

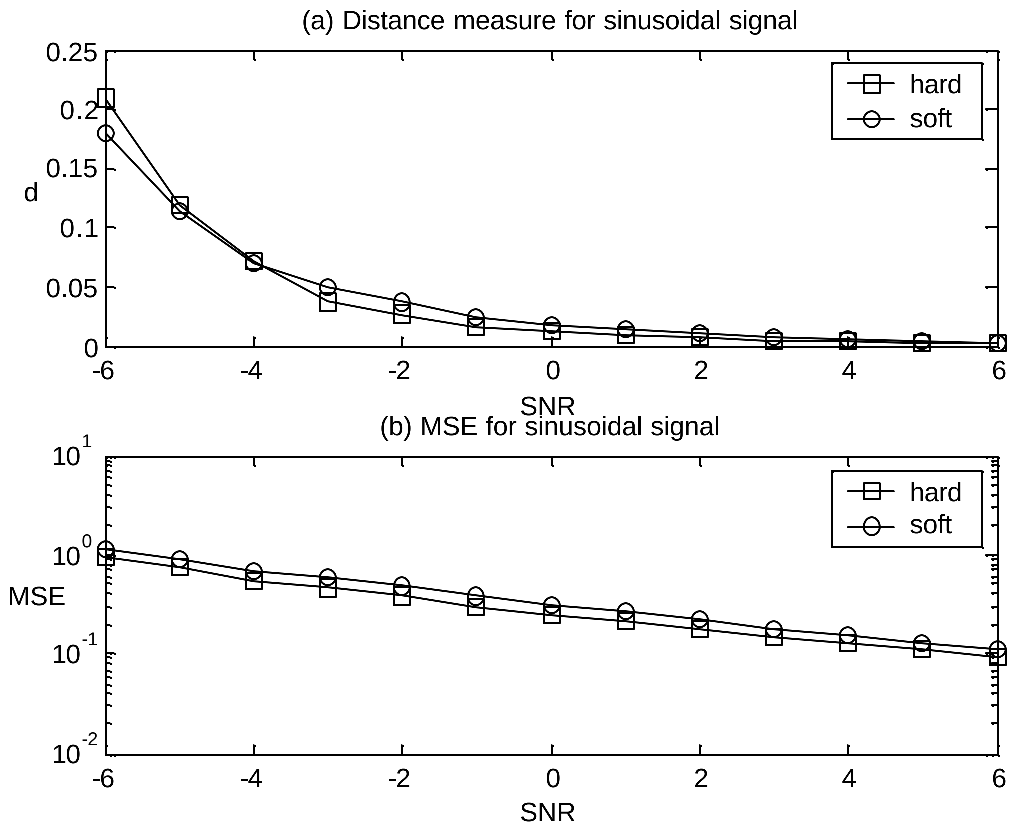

Figure 7. Distance measure *d* and MSE performance criteria for the sinusoidal signal; FFT-based denoising.

b. Constant Amplitude Chirp

The MSE results shown in Figure 8 indicate no significant differences in thresholding scheme performances, while the distance measure results indicate better performance for the soft thresholding scheme at low and medium SNR levels.

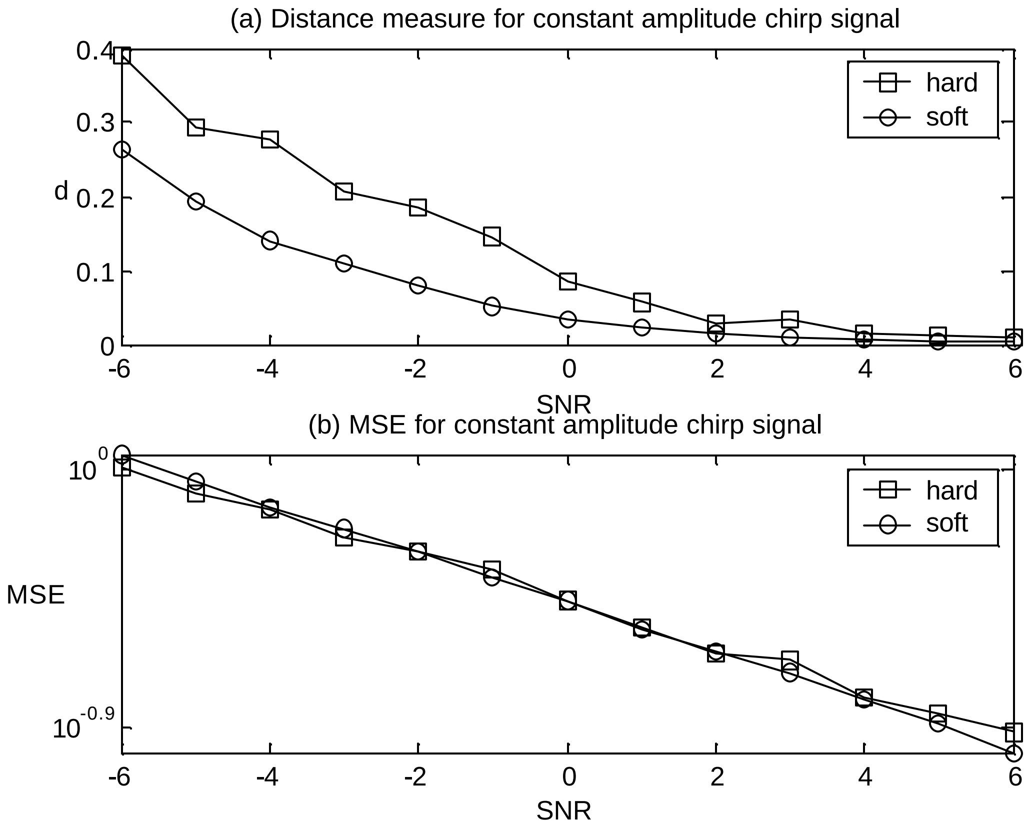

Figure 8. Distance measure *d* and MSE performance criteria for constant amplitude chirp signal; FFT-based denoising.

c. Chirp With an RC Time Constant-Like Amplitude Increase

Results shown in Figure 9 indicate that both thresholding schemes have similar MSE performances for SNR values below –3 dB, and that the soft thresholding scheme performs better for SNR values above –3 dB. The distance measure criterion indicates a better performance for soft thresholding for all SNR levels investigated.

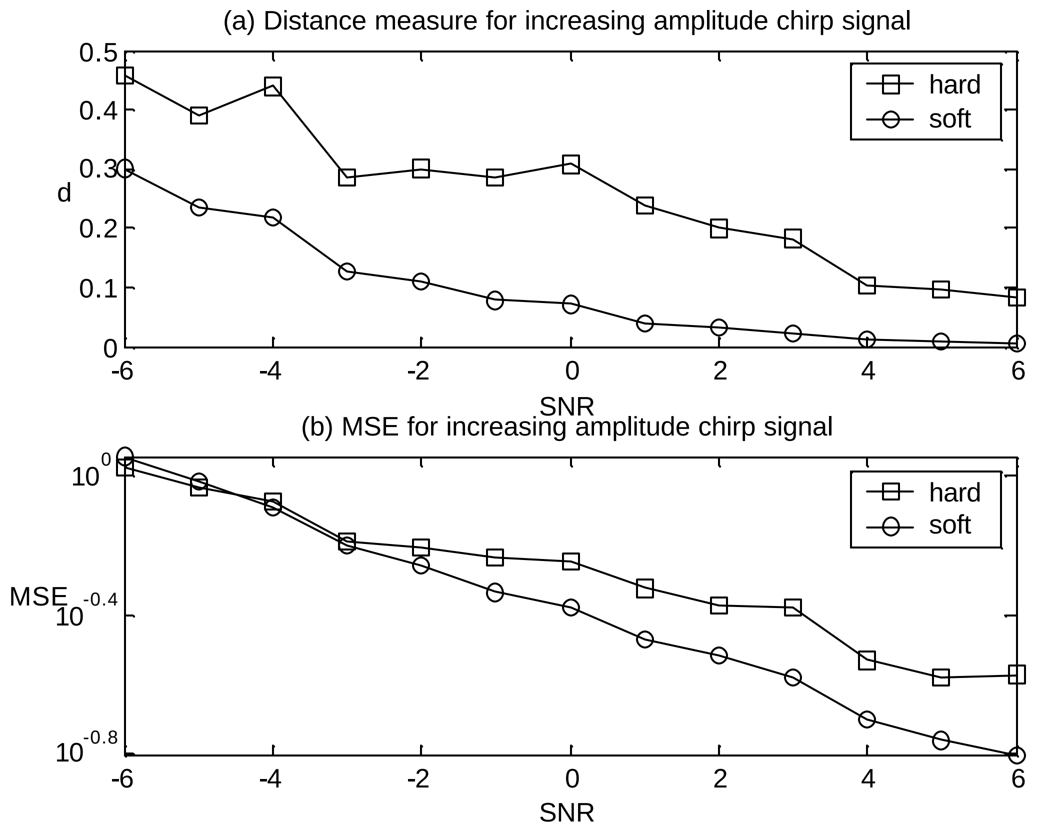

Figure 9. Distance measure *d* and MSE performance criteria for increasing amplitude chirp signal; FFT-based denoising.

2. Wavelet Transform-Based Denoising

Many parameters affect the performance of the wavelet transform-based denoising scheme. Recall that we consider four thresholding implementation schemes on the approximation coefficients: approximation coefficients left unperturbed (method 1), approximation coefficients thresholded identical to the detail coefficients (method 2), approximation coefficients median filtered with a filter of length three (method 3), and using an order-two predictor (method 4). Methods 3 and 4 do not perform as well as the first two methods because they oversmooth the approximation coefficients sequence, causing loss of signal power and degradation in the performance of the denoising scheme. Other parameters are the number of decomposition levels and the order of the Daubechies wavelet to be used (3, 4 and 5). The numbers in the figure legends next to the thresholding types are the Daubechies wavelet orders. MSE values obtained for all schemes are close to each other and do not provide much information about relative performances. As a result, we consider the distance measure *d* to compare performances. In addition, we only show results obtained for the best of the 4 methods for the approximation coefficients and the best of the 3 wavelet based decompositions (3, 4 or 5 level decompositions). Examples for noisy signals and their recovered versions obtained with WT-based denoising schemes are included in Appendix B.

a. Sinusoidal Signal

Results show that best performances are obtained with method 2 for this signal type. Figure 10 presents the WT-based scheme performance obtained for a 4-level decomposition with different wavelet orders and both thresholding schemes. Note that the sinusoidal signal has a constant frequency remaining in one of the frequency bands formed by the wavelet transform for the whole signal duration. Results show that the performance of the WT-based denoising scheme depends on the frequency of the specific test signal and the level of wavelet decomposition. Note that test signal frequencies were picked randomly for each trial. It should also be noted that when using method 1, the signal is left untouched only when it is located in the lowest frequency band (i.e., the band containing the approximation coefficients). However, noise when present is also left untouched in that lowest frequency band, while no such distinction is present in method 2.

Simulation results show that the soft thresholding scheme outperforms the hard thresholding implementation at SNR values below 2 dB, while all thresholding schemes perform similarly for higher SNRs.

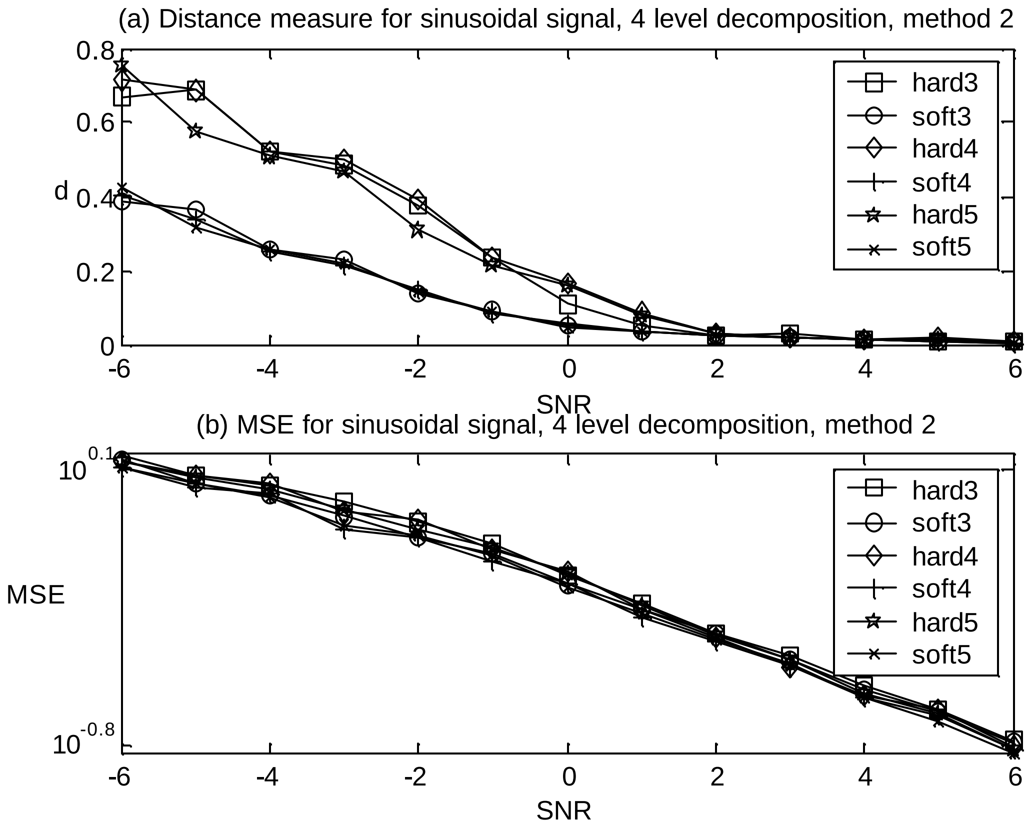

Figure 10. Distance measure and MSE for test signal 1, using a Wavelet-based 4-level decomposition and method 2, with Daubechies wavelet orders 3, 4 and 5.

b. Constant Amplitude Chirp

Results for the WT-based denoising scheme using 4-level decomposition and method 1 are presented in Figure 11. Method 1 was shown here because simulations indicated that method 1 has significantly higher performance than method 2 on the constant amplitude chirp signal. This result was to be expected as this signal has most of its power at low frequencies. Frequency components exist in several frequency bands, including the one containing the approximation coefficients. When method 2 is used, the sequence of approximation coefficients is tested for Gaussianity and may be estimated as Gaussian when the signal components are very short, resulting in thresholding of the signal components. However, such a problem does not exist in method 1 where the approximation coefficients are left untouched. Finally, simulation results show that a four-level decomposition performs better than a three or five-level decomposition.

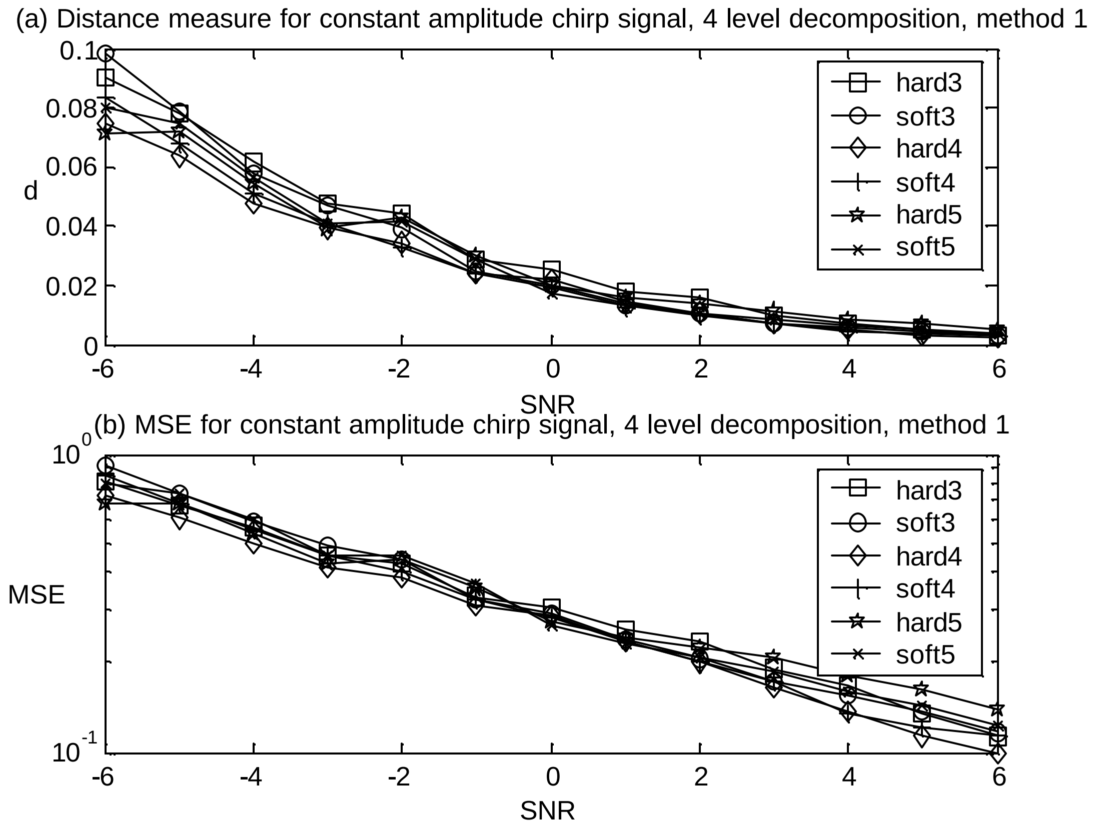

Figure 11. Performance for constant amplitude chirp signal, using a Wavelet based 4-level decomposition and method 1, with Daubechies wavelet orders 3, 4 and 5.

c. Chirp With an RC Time Constant-Like Amplitude Increase

Simulations show method 1 has the best performance of the 4 methods investigated and associated results are presented in Figure 12. Results shown in Figure 12 indicate that similar performances are obtained for all thresholding schemes and wavelet orders considered.

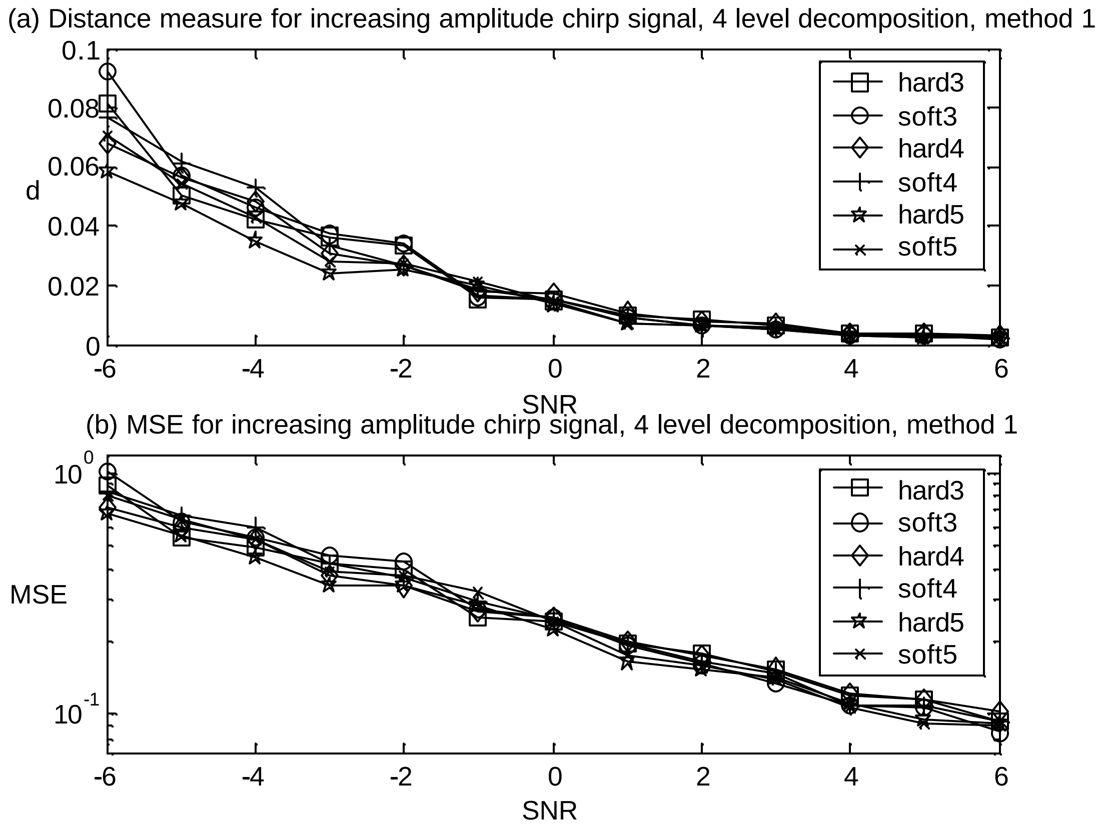

Figure 12. Performance for increasing amplitude chirp signal, using a Wavelet-based 4-level decomposition and method 1, with Daubechies wavelet orders 3, 4 and 5.

In this chapter the test signals were described and the simulation results were summarized. The conclusions are presented in the next chapter.

V. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

This work considered a Bootstrap-based denoising scheme to denoise deterministic signals embedded in additive Gaussian noise. The Bootstrap is a technique derived for improving the accuracy of a parameter estimator, used especially in situations where conventional techniques are not valid since the data considered is short in length. This technique is used for parameter estimation and hypothesis testing in our work.

The proposed Bootstrap-based denoising scheme has four steps. First, the noisy signal is transformed into a two-dimensional domain, using the Short-Time Fast Fourier or the Wavelet Transform. Next, Gaussianity tests are performed on the transformed data on a frequency band basis. No processing is applied to the data when the tested data is found to be non-Gaussian, as this indicates signal components in the data are dominant over Gaussian noise components. However, denoising is applied when the data tested is found to be Gaussian, which indicates that the noise components are dominant. Denoising is obtained by thresholding the frequency components in specific frequency bands to minimize noise effects. Finally, the recovered signal is obtained by applying the appropriate inverse transform.

Denoising scheme performances were investigated using three test signals: a sinusoid, a constant amplitude chirp and a chirp with increasing amplitude. MSE and cross-correlation measures were selected to investigate the relative performances of all four denoising schemes considered in this work. In most cases, for a given SNR the MSE values obtained for schemes with different decomposition types and levels were similar at all SNRs and were not useful in discriminating between the various schemes investigated. However, the distance measure was more sensitive and showed more discriminating information. Results show that FFT-based schemes perform better than WT-based schemes on the stationary sinusoid signal type, whereas WT-based scheme outperforms the FFT-based scheme on the chirp signal types considered. In the FFT case, soft and hard thresholding schemes perform similarly on the sinusoid, while soft thresholding outperforms hard thresholding on the chirp signals. Results also show that soft thresholding performs better on the stationary sinusoidal signal type but performs similar to hard thresholding on the chirps in the WT case. Note that thresholding methods are performed on the detail coefficients in the WT-based denoising case. However, using them directly on the approximation coefficients may cause signal loss. Therefore, four schemes were considered in the simulations: Leaving the approximation coefficients untouched, using the same thresholding schemes as the detail coefficients, using a median filter of order 3 on the approximation coefficients, and using a predictor of order 2 on the approximation coefficients. The second method performs best for the sinusoidal test signal whereas the first method performs the best for the chirp type test signals.

Daubechies wavelets of orders 3, 4 and 5 were selected in the WT-based denoising scheme. Simulations show that wavelets of this family with higher orders did not improve performance for signals with the given length.

The proposed denoising scheme was compared with the Wavelet shrinkage algorithm, which was introduced by Donoho and Johnstone [16, 17] to denoise signals embedded in additive white Gaussian noise with unit variance. Results indicate that the proposed WT-based denoising scheme performs better than the wavelet shrinkage algorithm for the sinusoidal test signal. However, the wavelet shrinkage algorithm outperforms the proposed scheme for the chirp type test signals. It should be noted that these results (Donoho and Johnstone) are restricted to the case of noise with unit variance.

B. FUTURE WORK

Our study was restricted to one wavelet family (Daubechies) only and further investigations should consider other types of wavelets to investigate the impact of the specific wavelet type on the resulting denoising performances.

APPENDIX A. MATLAB CODES

MATLAB simulations were performed to were used to test the performance of the denoising scheme. The codes used are presented in this chapter. The first and second sections present the codes for the FFT- and WT-based denoising schemes, respectively. The third section presents the sub-functions that were used in both of these codes.

A. FFT-BASED DENOISING

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          %
% testkurtv7: implements the FFT-based denoising scheme on %
%             three test signals                           %
%                                                          %
% SYNTAX    : testkurtv7                                   %
%                                                          %
% INPUT     : none                                         %
%                                                          %
% OUTPUT    : simulation results saved on disk             %
%                                                          %
% SUB FUNC  : MakeSignal.m                                 %
%             kurtosistest.m                               %
%                                                          %
% Written by Hasan E. KAN                                  %
%                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all

t=1:512;
ampl=sqrt(10.^([.6 0 -.6]));

kurlim=4.5;

for sigtype=1:3
     for trial=1:100
        recksoft=zeros(length(ampl),512);
        reckhard=zeros(length(ampl),512);
        if sigtype==1
           s=sin(2*pi*t*(1+ceil(rand*240))/512);
s=s/sqrt(mean(s.^2));
           sigtypes='sin';
        elseif sigtype==2
           s=sin(34.7./[.01:.01:612]);
s=s(41:552);s=s/sqrt(mean(s.^2));
           sigtypes='cch';
        elseif sigtype==3
              s=makesignal('Doppler',521);
s=s(10:521);s=s/sqrt(mean(s.^2));
              sigtypes='ech';
           end
           signal=ones(length(ampl),1)*s;
           sigpwr=sum(signal.^2,2)/512;

           n=randn(length(ampl),512);
           noise=((ampl./std(n'))'*ones(1,512)).*n;
           noipwr=sum(noise.^2,2)/512;

           SNR=10*log10(sigpwr./noipwr);

           x=signal+noise;

        for snr=1:length(ampl)%1:21
           fx=zeros(61,32);
           for segmentno=1:61
              fx(segmentno,:)=fft(x(snr,(segmentno-
1)*8+1:(segmentno-1)*8+32).*triang(32)');
           end
           fksoft=zeros(61,32);
           fksoftr=zeros(61,17);
           fksofti=zeros(61,17);
           fkhard=zeros(61,32);
           fkhardr=zeros(61,17);
           fkhardi=zeros(61,17);
           kurr=zeros(length(ampl),17);
           kuri=zeros(length(ampl),17);
           for segmentno=1:17

[fksoftr(:,segmentno),fkhardr(:,segmentno),kurr(segmentno)]=kurto
sistest(real(fx(:,segmentno)),kurlim);
              if imag(fx(:,segmentno))~=0

[fksofti(:,segmentno),fkhardi(:,segmentno),kuri(segmentno)]=kurto
sistest(imag(fx(:,segmentno)),kurlim);
              end
           end
           fksoft(:,1:17)=fksoftr(:,1:17)+j*fksofti(:,1:17);
           fkhard(:,1:17)=fkhardr(:,1:17)+j*fkhardi(:,1:17);
           fksoft(:,18:32)=conj(fliplr(fksoft(:,2:16)));
           fkhard(:,18:32)=conj(fliplr(fkhard(:,2:16)));
           for segmentno=1:61
              recksoft(snr,(segmentno-1)*8+1:(segmentno-
1)*8+32)=recksoft(snr,(segmentno-1)*8+1:(segmentno-
1)*8+32)+ifft(fksoft(segmentno,:));
```

```
                reckhard(snr,(segmentno-1)*8+1:(segmentno-
1)*8+32)=reckhard(snr,(segmentno-1)*8+1:(segmentno-
1)*8+32)+ifft(fkhard(segmentno,:));
            end
        end
        recksoft=recksoft*0.5;
        reckhard=reckhard*0.5;
        msesoft(:,trial)=sum((signal-recksoft).^2,2)/512;
        msehard(:,trial)=sum((signal-reckhard).^2,2)/512;
        for snr=1:21

corsoft(snr,trial)=xcorr(s,recksoft(snr,:),0,'coeff');

corhard(snr,trial)=xcorr(s,reckhard(snr,:),0,'coeff');
        end
    end
    save(sprintf('mse%s',sigtypes),'msemed','msesoft','msehard')
    save(sprintf('cor%s',sigtypes),'corsoft','corhard')
end
```

B. WT-BASED DENOISING

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          %
% testkurtv7: implements the WT-based denoising scheme on  %
%             three test signals                           %
%                                                          %
% SYNTAX    : testwav2                                     %
%                                                          %
% INPUT     : none                                         %
%                                                          %
% OUTPUT    : simulation results saved on disk             %
%                                                          %
% SUB FUNC  : MakeSignal.m                                 %
%             kurtosistest.m                               %
%                                                          %
% Written by Hasan E. KAN                                  %
%                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all

t=1:512;
ampl=sqrt(10.^([.6 0 -.6]));

kurlim=4.5;

rec5db3soft=zeros(length(ampl),512,4);
rec5db3hard=zeros(length(ampl),512,4);
```

```
rec4db3soft=zeros(length(ampl),512,4);
rec4db3hard=zeros(length(ampl),512,4);
rec3db3soft=zeros(length(ampl),512,4);
rec3db3hard=zeros(length(ampl),512,4);

rec5db4soft=zeros(length(ampl),512,4);
rec5db4hard=zeros(length(ampl),512,4);
rec4db4soft=zeros(length(ampl),512,4);
rec4db4hard=zeros(length(ampl),512,4);
rec3db4soft=zeros(length(ampl),512,4);
rec3db4hard=zeros(length(ampl),512,4);

rec5db5soft=zeros(length(ampl),512,4);
rec5db5hard=zeros(length(ampl),512,4);
rec4db5soft=zeros(length(ampl),512,4);
rec4db5hard=zeros(length(ampl),512,4);
rec3db5soft=zeros(length(ampl),512,4);
rec3db5hard=zeros(length(ampl),512,4);

for sigtype=1:3
    for trial=1:100
        if sigtype==1

s=sin(2*pi*t*(1+ceil(rand*240))/512);s=s/sqrt(sum(s.^2)/512);
            sigtypes='sin';
        elseif sigtype==2

s=sin(34.7./[.01:.01:612]);s=s(41:552);s=s/sqrt(sum(s.^2)/512);
            sigtypes='cch';
        elseif sigtype==3

s=makesignal('Doppler',521);s=s(10:521);s=s/sqrt(sum(s.^2)/512);
            sigtypes='ech';
        end
           signal=ones(length(ampl),1)*s;
           sigpwr=sum(signal.^2,2)/512;

           n=randn(length(ampl),512);
           noise=((ampl./std(n'))'*ones(1,512)).*n;
           noipwr=sum(noise.^2,2)/512;

           SNR=10*log10(sigpwr./noipwr);

           x=signal+noise;

        for snr=1:length(ampl)
            [cdb3(snr,:),ldb3]=wavedec(x(snr,:),3,'db3');
            a3db3(snr,:)=appcoef(cdb3(snr,:),ldb3,'db3',3);

[d1db3(snr,:),d2db3(snr,:),d3db3(snr,:)]=detcoef(cdb3(snr,:),ldb3
,[1,2,3]);
            [a4db3(snr,:),d4db3(snr,:)]=dwt(a3db3(snr,:),'db3');
            [a5db3(snr,:),d5db3(snr,:)]=dwt(a4db3(snr,:),'db3');

            [cdb4(snr,:),ldb4]=wavedec(x(snr,:),3,'db4');
            a3db4(snr,:)=appcoef(cdb4(snr,:),ldb4,'db4',3);

[d1db4(snr,:),d2db4(snr,:),d3db4(snr,:)]=detcoef(cdb4(snr,:),ldb4
,[1,2,3]);
            [a4db4(snr,:),d4db4(snr,:)]=dwt(a3db4(snr,:),'db4');
            [a5db4(snr,:),d5db4(snr,:)]=dwt(a4db4(snr,:),'db4');

            [cdb5(snr,:),ldb5]=wavedec(x(snr,:),3,'db5');
            a3db5(snr,:)=appcoef(cdb5(snr,:),ldb5,'db5',3);

[d1db5(snr,:),d2db5(snr,:),d3db5(snr,:)]=detcoef(cdb5(snr,:),ldb5
,[1,2,3]);
            [a4db5(snr,:),d4db5(snr,:)]=dwt(a3db5(snr,:),'db5');
            [a5db5(snr,:),d5db5(snr,:)]=dwt(a4db5(snr,:),'db5');
        end

ra5db3soft=zeros(length(ampl),length(a5db3));ra5db3hard=zeros(len
gth(ampl),length(a5db3));

ra4db3soft=zeros(length(ampl),length(a4db3));ra4db3hard=zeros(len
gth(ampl),length(a4db3));

ra3db3soft=zeros(length(ampl),length(a3db3));ra3db3hard=zeros(len
gth(ampl),length(a3db3));

rd5db3soft=zeros(length(ampl),length(d5db3(1,:)));rd5db3hard=zero
s(length(ampl),length(d5db3(1,:)));

rd4db3soft=zeros(length(ampl),length(d4db3));rd4db3hard=zeros(len
gth(ampl),length(d4db3));

rd3db3soft=zeros(length(ampl),length(d3db3));rd3db3hard=zeros(len
gth(ampl),length(d3db3));

rd2db3soft=zeros(length(ampl),length(d2db3));rd2db3hard=zeros(len
gth(ampl),length(d2db3));

rd1db3soft=zeros(length(ampl),length(d1db3));rd1db3hard=zeros(len
gth(ampl),length(d1db3));


ra5db4soft=zeros(length(ampl),length(a5db4));ra5db4hard=zeros(len
gth(ampl),length(a5db4));

ra4db4soft=zeros(length(ampl),length(a4db4));ra4db4hard=zeros(len
gth(ampl),length(a4db4));

ra3db4soft=zeros(length(ampl),length(a3db4));ra3db4hard=zeros(len
gth(ampl),length(a3db4));
```

```
rd5db4soft=zeros(length(ampl),length(d5db4));rd5db4hard=zeros(len
gth(ampl),length(d5db4));

rd4db4soft=zeros(length(ampl),length(d4db4));rd4db4hard=zeros(len
gth(ampl),length(d4db4));

rd3db4soft=zeros(length(ampl),length(d3db4));rd3db4hard=zeros(len
gth(ampl),length(d3db4));

rd2db4soft=zeros(length(ampl),length(d2db4));rd2db4hard=zeros(len
gth(ampl),length(d2db4));

rd1db4soft=zeros(length(ampl),length(d1db4));rd1db4hard=zeros(len
gth(ampl),length(d1db4));


ra5db5soft=zeros(length(ampl),length(a5db5));ra5db5hard=zeros(len
gth(ampl),length(a5db5));

ra4db5soft=zeros(length(ampl),length(a4db5));ra4db5hard=zeros(len
gth(ampl),length(a4db5));

ra3db5soft=zeros(length(ampl),length(a3db5));ra3db5hard=zeros(len
gth(ampl),length(a3db5));

rd5db5soft=zeros(length(ampl),length(d5db5));rd5db5hard=zeros(len
gth(ampl),length(d5db5));

rd4db5soft=zeros(length(ampl),length(d4db5));rd4db5hard=zeros(len
gth(ampl),length(d4db5));

rd3db5soft=zeros(length(ampl),length(d3db5));rd3db5hard=zeros(len
gth(ampl),length(d3db5));

rd2db5soft=zeros(length(ampl),length(d2db5));rd2db5hard=zeros(len
gth(ampl),length(d2db5));

rd1db5soft=zeros(length(ampl),length(d1db5));rd1db5hard=zeros(len
gth(ampl),length(d1db5));
        for method=1:4
                if method==1
                ra5db3soft=a5db3;ra5db3hard=a5db3;
                ra4db3soft=a4db3;ra4db3hard=a4db3;
                ra3db3soft=a3db3;ra3db3hard=a3db3;
                ra5db4soft=a5db4;ra5db4hard=a5db4;
                ra4db4soft=a4db4;ra4db4hard=a4db4;
                ra3db4soft=a3db4;ra3db4hard=a3db4;
                ra5db5soft=a5db5;ra5db5hard=a5db5;
                ra4db5soft=a4db5;ra4db5hard=a4db5;
                ra3db5soft=a3db5;ra3db5hard=a3db5;
                method1='no operation on approximation coeff.';
                elseif method==2
                for snr=1:length(ampl)

[ra5db3soft(snr,:),ra5db3hard(snr,:),kura5db3(snr)]=kurtosistest(
a5db3(snr,:),kurlim);

[ra4db3soft(snr,:),ra4db3hard(snr,:),kura4db3(snr)]=kurtosistest(
a4db3(snr,:),kurlim);

[ra3db3soft(snr,:),ra3db3hard(snr,:),kura3db3(snr)]=kurtosistest(
a3db3(snr,:),kurlim);

[ra5db4soft(snr,:),ra5db4hard(snr,:),kura5db4(snr)]=kurtosistest(
a5db4(snr,:),kurlim);

[ra4db4soft(snr,:),ra4db4hard(snr,:),kura4db4(snr)]=kurtosistest(
a4db4(snr,:),kurlim);

[ra3db4soft(snr,:),ra3db4hard(snr,:),kura3db4(snr)]=kurtosistest(
a3db4(snr,:),kurlim);

[ra5db5soft(snr,:),ra5db5hard(snr,:),kura5db5(snr)]=kurtosistest(
a5db5(snr,:),kurlim);

[ra4db5soft(snr,:),ra4db5hard(snr,:),kura4db5(snr)]=kurtosistest(
a4db5(snr,:),kurlim);

[ra3db5soft(snr,:),ra3db5hard(snr,:),kura3db5(snr)]=kurtosistest(
a3db5(snr,:),kurlim);
                end
                method1='using   threshold   on   approximation
coeff.';
                elseif method==3

ra5db3soft=medfilt1(a5db3',3)';ra5db3hard=ra5db3soft;

ra4db3soft=medfilt1(a4db3',3)';ra4db3hard=ra4db3soft;

ra3db3soft=medfilt1(a3db3',3)';ra3db3hard=ra3db3soft;

ra5db4soft=medfilt1(a5db4',3)';ra5db4hard=ra5db4soft;

ra4db4soft=medfilt1(a4db4',3)';ra4db4hard=ra4db4soft;

ra3db4soft=medfilt1(a3db4',3)';ra3db4hard=ra3db4soft;

ra5db5soft=medfilt1(a5db5',3)';ra5db5hard=ra5db5soft;

ra4db5soft=medfilt1(a4db5',3)';ra4db5hard=ra4db5soft;

ra3db5soft=medfilt1(a3db5',3)';ra3db5hard=ra3db5soft;
                method1='using medfilt3 on approximation coeff.';
                else

rra5db3=xcorr(a5db3',2,'unbiased');rra5db3=rra5db3(3:5,4*[0:2]+1)
;

rra4db3=xcorr(a4db3',2,'unbiased');rra4db3=rra4db3(3:5,4*[0:2]+1)
;

rra3db3=xcorr(a3db3',2,'unbiased');rra3db3=rra3db3(3:5,4*[0:2]+1)
;

rra5db4=xcorr(a5db4',2,'unbiased');rra5db4=rra5db4(3:5,4*[0:2]+1)
;

rra4db4=xcorr(a4db4',2,'unbiased');rra4db4=rra4db4(3:5,4*[0:2]+1)
;

rra3db4=xcorr(a3db4',2,'unbiased');rra3db4=rra3db4(3:5,4*[0:2]+1)
;

rra5db5=xcorr(a5db5',2,'unbiased');rra5db5=rra5db5(3:5,4*[0:2]+1)
;

rra4db5=xcorr(a4db5',2,'unbiased');rra4db5=rra4db5(3:5,4*[0:2]+1)
;

rra3db5=xcorr(a3db5',2,'unbiased');rra3db5=rra3db5(3:5,4*[0:2]+1)
;
                for snr=1:length(ampl)

Rxa5db3=toeplitz(rra5db3(:,snr));Rxxa5db3=Rxa5db3(1:2,1:2);

Rxa4db3=toeplitz(rra4db3(:,snr));Rxxa4db3=Rxa4db3(1:2,1:2);

Rxa3db3=toeplitz(rra3db3(:,snr));Rxxa3db3=Rxa3db3(1:2,1:2);

Rxa5db4=toeplitz(rra5db4(:,snr));Rxxa5db4=Rxa5db4(1:2,1:2);

Rxa4db4=toeplitz(rra4db4(:,snr));Rxxa4db4=Rxa4db4(1:2,1:2);

Rxa3db4=toeplitz(rra3db4(:,snr));Rxxa3db4=Rxa3db4(1:2,1:2);

Rxa5db5=toeplitz(rra5db5(:,snr));Rxxa5db5=Rxa5db5(1:2,1:2);

Rxa4db5=toeplitz(rra4db5(:,snr));Rxxa4db5=Rxa4db5(1:2,1:2);

Rxa3db5=toeplitz(rra3db5(:,snr));Rxxa3db5=Rxa3db5(1:2,1:2);
                    aa5db3=Rxxa5db3\Rxa5db3(2:3,1);
                    aa4db3=Rxxa4db3\Rxa4db3(2:3,1);
                    aa3db3=Rxxa3db3\Rxa3db3(2:3,1);
                    aa5db4=Rxxa5db4\Rxa5db4(2:3,1);
                    aa4db4=Rxxa4db4\Rxa4db4(2:3,1);
                    aa3db4=Rxxa3db4\Rxa3db4(2:3,1);
                    aa5db5=Rxxa5db5\Rxa5db5(2:3,1);
                    aa4db5=Rxxa4db5\Rxa4db5(2:3,1);
                    aa3db5=Rxxa3db5\Rxa3db5(2:3,1);
                    ra5db3soft(snr,:)=filter([0;
aa5db3],1,a5db3(snr,:));
                    ra4db3soft(snr,:)=filter([0;
aa4db3],1,a4db3(snr,:));
                    ra3db3soft(snr,:)=filter([0;
aa3db3],1,a3db3(snr,:));
                    ra5db4soft(snr,:)=filter([0;
aa5db4],1,a5db4(snr,:));
                    ra4db4soft(snr,:)=filter([0;
aa4db4],1,a4db4(snr,:));
                    ra3db4soft(snr,:)=filter([0;
aa3db4],1,a3db4(snr,:));
                    ra5db5soft(snr,:)=filter([0;
aa5db5],1,a5db5(snr,:));
                    ra4db5soft(snr,:)=filter([0;
aa4db5],1,a4db5(snr,:));
                    ra3db5soft(snr,:)=filter([0;
aa3db5],1,a3db5(snr,:));
                end
                ra5db3soft(:,1:2)=a5db3(:,1:2);
                ra4db3soft(:,1:2)=a4db3(:,1:2);
                ra3db3soft(:,1:2)=a3db3(:,1:2);
                ra5db4soft(:,1:2)=a5db4(:,1:2);
                ra4db4soft(:,1:2)=a4db4(:,1:2);
                ra3db4soft(:,1:2)=a3db4(:,1:2);
                ra5db5soft(:,1:2)=a5db5(:,1:2);
                ra4db5soft(:,1:2)=a4db5(:,1:2);
                ra3db5soft(:,1:2)=a3db5(:,1:2);
                ra5db3hard=ra5db3soft;
                ra4db3hard=ra4db3soft;
                ra3db3hard=ra3db3soft;
                ra5db4hard=ra5db4soft;
                ra4db4hard=ra4db4soft;
                ra3db4hard=ra3db4soft;
                ra5db5hard=ra5db5soft;
                ra4db5hard=ra4db5soft;
                ra3db5hard=ra3db5soft;
                method1='prediction on approximation coeff.';
                end
                for snr=1:length(ampl)

[rd5db3soft(snr,:),rd5db3hard(snr,:),kurd5db3(snr)]=kurtosistest(
d5db3(snr,:),kurlim);

[rd4db3soft(snr,:),rd4db3hard(snr,:),kurd4db3(snr)]=kurtosistest(
d4db3(snr,:),kurlim);
```

```
[rd3db3soft(snr,:),rd3db3hard(snr,:),kurd3db3(snr)]=kurtosistest(
d3db3(snr,:),kurlim);

[rd2db3soft(snr,:),rd2db3hard(snr,:),kurd2db3(snr)]=kurtosistest(
d2db3(snr,:),kurlim);

[rd1db3soft(snr,:),rd1db3hard(snr,:),kurd1db3(snr)]=kurtosistest(
d1db3(snr,:),kurlim);


[rd5db4soft(snr,:),rd5db4hard(snr,:),kurd5db4(snr)]=kurtosistest(
d5db4(snr,:),kurlim);

[rd4db4soft(snr,:),rd4db4hard(snr,:),kurd4db4(snr)]=kurtosistest(
d4db4(snr,:),kurlim);

[rd3db4soft(snr,:),rd3db4hard(snr,:),kurd3db4(snr)]=kurtosistest(
d3db4(snr,:),kurlim);

[rd2db4soft(snr,:),rd2db4hard(snr,:),kurd2db4(snr)]=kurtosistest(
d2db4(snr,:),kurlim);

[rd1db4soft(snr,:),rd1db4hard(snr,:),kurd1db4(snr)]=kurtosistest(
d1db4(snr,:),kurlim);


[rd5db5soft(snr,:),rd5db5hard(snr,:),kurd5db5(snr)]=kurtosistest(
d5db5(snr,:),kurlim);

[rd4db5soft(snr,:),rd4db5hard(snr,:),kurd4db5(snr)]=kurtosistest(
d4db5(snr,:),kurlim);

[rd3db5soft(snr,:),rd3db5hard(snr,:),kurd3db5(snr)]=kurtosistest(
d3db5(snr,:),kurlim);

[rd2db5soft(snr,:),rd2db5hard(snr,:),kurd2db5(snr)]=kurtosistest(
d2db5(snr,:),kurlim);

[rd1db5soft(snr,:),rd1db5hard(snr,:),kurd1db5(snr)]=kurtosistest(
d1db5(snr,:),kurlim);
            end
                R5db3soft=[ra5db3soft   rd5db3soft   rd4db3soft
rd3db3soft rd2db3soft rd1db3soft];
                R5db3hard=[ra5db3hard   rd5db3hard   rd4db3hard
rd3db3hard rd2db3hard rd1db3hard];
                R4db3soft=[ra4db3soft   rd4db3soft   rd3db3soft
rd2db3soft rd1db3soft];
                R4db3hard=[ra4db3hard   rd4db3hard   rd3db3hard
rd2db3hard rd1db3hard];
                R3db3soft=[ra3db3soft   rd3db3soft   rd2db3soft
rd1db3soft];
```

```
                R3db3hard=[ra3db3hard   rd3db3hard   rd2db3hard
rd1db3hard];

                R5db4soft=[ra5db4soft   rd5db4soft   rd4db4soft
rd3db4soft rd2db4soft rd1db4soft];
                R5db4hard=[ra5db4hard   rd5db4hard   rd4db4hard
rd3db4hard rd2db4hard rd1db4hard];
                R4db4soft=[ra4db4soft   rd4db4soft   rd3db4soft
rd2db4soft rd1db4soft];
                R4db4hard=[ra4db4hard   rd4db4hard   rd3db4hard
rd2db4hard rd1db4hard];
                R3db4soft=[ra3db4soft   rd3db4soft   rd2db4soft
rd1db4soft];
                R3db4hard=[ra3db4hard   rd3db4hard   rd2db4hard
rd1db4hard];

                R5db5soft=[ra5db5soft   rd5db5soft   rd4db5soft
rd3db5soft rd2db5soft rd1db5soft];
                R5db5hard=[ra5db5hard   rd5db5hard   rd4db5hard
rd3db5hard rd2db5hard rd1db5hard];
                R4db5soft=[ra4db5soft   rd4db5soft   rd3db5soft
rd2db5soft rd1db5soft];
                R4db5hard=[ra4db5hard   rd4db5hard   rd3db5hard
rd2db5hard rd1db5hard];
                R3db5soft=[ra3db5soft   rd3db5soft   rd2db5soft
rd1db5soft];
                R3db5hard=[ra3db5hard   rd3db5hard   rd2db5hard
rd1db5hard];
            for snr=1:length(ampl)

rec5db3soft(snr,:,method)=waverec(R5db3soft(snr,:),[20 20 36 68
131 258 512],'db3');

     rec5db3hard(snr,:,method)=waverec(R5db3hard(snr,:),[20   20
36 68 131 258 512],'db3');

rec4db3soft(snr,:,method)=waverec(R4db3soft(snr,:),[36 36 68 131
258 512],'db3');

     rec4db3hard(snr,:,method)=waverec(R4db3hard(snr,:),[36   36
68 131 258 512],'db3');

rec3db3soft(snr,:,method)=waverec(R3db3soft(snr,:),[68 68 131 258
512],'db3');

     rec3db3hard(snr,:,method)=waverec(R3db3hard(snr,:),[68   68
131 258 512],'db3');


rec5db4soft(snr,:,method)=waverec(R5db4soft(snr,:),[22 22 38 70
133 259 512],'db4');
```

```
     rec5db4hard(snr,:,method)=waverec(R5db4hard(snr,:),[22   22
38 70 133 259 512],'db4');

rec4db4soft(snr,:,method)=waverec(R4db4soft(snr,:),[38 38 70 133
259 512],'db4');

     rec4db4hard(snr,:,method)=waverec(R4db4hard(snr,:),[38   38
70 133 259 512],'db4');

rec3db4soft(snr,:,method)=waverec(R3db4soft(snr,:),[70 70 133 259
512],'db4');

     rec3db4hard(snr,:,method)=waverec(R3db4hard(snr,:),[70   70
133 259 512],'db4');


rec5db5soft(snr,:,method)=waverec(R5db5soft(snr,:),[24 24 40 71
134 260 512],'db5');

     rec5db5hard(snr,:,method)=waverec(R5db5hard(snr,:),[24   24
40 71 134 260 512],'db5');

rec4db5soft(snr,:,method)=waverec(R4db5soft(snr,:),[40 40 71 134
260 512],'db5');

     rec4db5hard(snr,:,method)=waverec(R4db5hard(snr,:),[40   40
71 134 260 512],'db5');

rec3db5soft(snr,:,method)=waverec(R3db5soft(snr,:),[71 71 134 260
512],'db5');

     rec3db5hard(snr,:,method)=waverec(R3db5hard(snr,:),[71   71
134 260 512],'db5');
            end
        end
        signa=repmat(signal,[1,1,4]);

          mse5db3soft(:,trial,:)=sum((signa-
rec5db3soft).^2,2)/512;
          mse5db3hard(:,trial,:)=sum((signa-
rec5db3hard).^2,2)/512;
          mse4db3soft(:,trial,:)=sum((signa-
rec4db3soft).^2,2)/512;
          mse4db3hard(:,trial,:)=sum((signa-
rec4db3hard).^2,2)/512;
          mse3db3soft(:,trial,:)=sum((signa-
rec3db3soft).^2,2)/512;
          mse3db3hard(:,trial,:)=sum((signa-
rec3db3hard).^2,2)/512;
```

```
            mse5db4soft(:,trial,:)=sum((signa-
rec5db4soft).^2,2)/512;
            mse5db4hard(:,trial,:)=sum((signa-
rec5db4hard).^2,2)/512;
            mse4db4soft(:,trial,:)=sum((signa-
rec4db4soft).^2,2)/512;
            mse4db4hard(:,trial,:)=sum((signa-
rec4db4hard).^2,2)/512;
            mse3db4soft(:,trial,:)=sum((signa-
rec3db4soft).^2,2)/512;
            mse3db4hard(:,trial,:)=sum((signa-
rec3db4hard).^2,2)/512;

            mse5db5soft(:,trial,:)=sum((signa-
rec5db5soft).^2,2)/512;
            mse5db5hard(:,trial,:)=sum((signa-
rec5db5hard).^2,2)/512;
            mse4db5soft(:,trial,:)=sum((signa-
rec4db5soft).^2,2)/512;
            mse4db5hard(:,trial,:)=sum((signa-
rec4db5hard).^2,2)/512;
            mse3db5soft(:,trial,:)=sum((signa-
rec3db5soft).^2,2)/512;
            mse3db5hard(:,trial,:)=sum((signa-
rec3db5hard).^2,2)/512;
        for snr=1:21
            for method=1:4

cor3db3soft(snr,trial,method)=xcorr(s,rec3db3soft(snr,:,method),0
,'coeff');

cor3db4soft(snr,trial,method)=xcorr(s,rec3db4soft(snr,:,method),0
,'coeff');

cor3db5soft(snr,trial,method)=xcorr(s,rec3db5soft(snr,:,method),0
,'coeff');

cor4db3soft(snr,trial,method)=xcorr(s,rec4db3soft(snr,:,method),0
,'coeff');

cor4db4soft(snr,trial,method)=xcorr(s,rec4db4soft(snr,:,method),0
,'coeff');

cor4db5soft(snr,trial,method)=xcorr(s,rec4db5soft(snr,:,method),0
,'coeff');

cor5db3soft(snr,trial,method)=xcorr(s,rec5db3soft(snr,:,method),0
,'coeff');

cor5db4soft(snr,trial,method)=xcorr(s,rec5db4soft(snr,:,method),0
,'coeff');
```

```
cor5db5soft(snr,trial,method)=xcorr(s,rec5db5soft(snr,:,method),0
,'coeff');

cor3db3hard(snr,trial,method)=xcorr(s,rec3db3hard(snr,:,method),0
,'coeff');

cor3db4hard(snr,trial,method)=xcorr(s,rec3db4hard(snr,:,method),0
,'coeff');

cor3db5hard(snr,trial,method)=xcorr(s,rec3db5hard(snr,:,method),0
,'coeff');

cor4db3hard(snr,trial,method)=xcorr(s,rec4db3hard(snr,:,method),0
,'coeff');

cor4db4hard(snr,trial,method)=xcorr(s,rec4db4hard(snr,:,method),0
,'coeff');

cor4db5hard(snr,trial,method)=xcorr(s,rec4db5hard(snr,:,method),0
,'coeff');

cor5db3hard(snr,trial,method)=xcorr(s,rec5db3hard(snr,:,method),0
,'coeff');

cor5db4hard(snr,trial,method)=xcorr(s,rec5db4hard(snr,:,method),0
,'coeff');

cor5db5hard(snr,trial,method)=xcorr(s,rec5db5hard(snr,:,method),0
,'coeff');
            end
        end
    end

save(sprintf('mse%s3lvl',sigtypes),'msemed','mse3db3soft','mse3db
3hard','mse3db4soft','mse3db4hard','mse3db5soft','mse3db5hard')

save(sprintf('mse%s4lvl',sigtypes),'msemed','mse4db3soft','mse4db
3hard','mse4db4soft','mse4db4hard','mse4db5soft','mse4db5hard')

save(sprintf('mse%s5lvl',sigtypes),'msemed','mse5db3soft','mse5db
3hard','mse5db4soft','mse5db4hard','mse5db5soft','mse5db5hard')

save(sprintf('cor%s3lvl',sigtypes),'cor3db3soft','cor3db3hard','c
or3db4soft','cor3db4hard','cor3db5soft','cor3db5hard')

save(sprintf('cor%s4lvl',sigtypes),'cor4db3soft','cor4db3hard','c
or4db4soft','cor4db4hard','cor4db5soft','cor4db5hard')

save(sprintf('cor%s5lvl',sigtypes),'cor5db3soft','cor5db3hard','c
or5db4soft','cor5db4hard','cor5db5soft','cor5db5hard')
end
```

C. SUB FUNCTIONS

1. MakeSignal

This code is part of Wavelab version 802 [11].

```
function sig = MakeSignal(Name,n)
% MakeSignal -- Make artificial signal
%  Usage
%    sig = MakeSignal(Name,n)
%  Inputs
%    Name   string: 'HeaviSine', 'Bumps', 'Blocks',
%             'Doppler', 'Ramp', 'Cusp', 'Sing', 'HiSine',
%             'LoSine', 'LinChirp', 'TwoChirp', 'QuadChirp',
%             'MishMash', 'WernerSorrows' (Heisenberg),
%             'Leopold' (Kronecker), 'Piece-Regular' (Piece-Wise
Smooth),
%          'Riemann','HypChirps','LinChirps', 'Chirps', 'Gabor'
%          'sineoneoverx','Cusp2','SmoothCusp','Gaussian'
%          'Piece-Polynomial' (Piece-Wise 3rd degree polynomial)
%    n      desired signal length
%  Outputs
%    sig    1-d signal
%
%  References
%    Various articles of D.L. Donoho and I.M. Johnstone
%
   if nargin > 1,
     t = (1:n) ./n;
   end
     if strcmp(Name,'HeaviSine'),
         sig = 4.*sin(4*pi.*t);
         sig = sig - sign(t - .3) - sign(.72 - t);
     elseif strcmp(Name,'Bumps'),
         pos = [ .1 .13 .15 .23 .25 .40 .44 .65  .76 .78 .81];
         hgt = [ 4  5   3   4  5  4.2 2.1 4.3  3.1 5.1 4.2];
         wth = [.005 .005 .006 .01 .01 .03 .01 .01  .005 .008
.005];
         sig = zeros(size(t));
         for j =1:length(pos)
            sig = sig + hgt(j)./( 1 + abs((t -
pos(j))./wth(j))).^4;
         end
     elseif strcmp(Name,'Blocks'),
         pos = [ .1 .13 .15 .23 .25 .40 .44 .65  .76 .78 .81];
         hgt = [4 (-5) 3 (-4) 5 (-4.2) 2.1 4.3  (-3.1) 2.1 (-
4.2)];
         sig = zeros(size(t));
         for j=1:length(pos)
             sig = sig + (1 + sign(t-pos(j))).*(hgt(j)/2) ;
         end
     elseif strcmp(Name,'Doppler'),
         sig = sqrt(t.*(1-t)).*sin((2*pi*1.05) ./(t+.05));
     elseif strcmp(Name,'Ramp'),
         sig = t - (t >= .37);
     elseif strcmp(Name,'Cusp'),
         sig = sqrt(abs(t - .37));
     elseif strcmp(Name,'Sing'),
         k = floor(n * .37);
         sig = 1 ./abs(t - (k+.5)/n);
     elseif strcmp(Name,'HiSine'),
         sig = sin( pi * (n * .6902) .* t);
     elseif strcmp(Name,'LoSine'),
         sig = sin( pi * (n * .3333) .* t);
     elseif strcmp(Name,'LinChirp'),
         sig = sin(pi .* t .* ((n .* .500) .* t));
     elseif strcmp(Name,'TwoChirp'),
         sig = sin(pi .* t .* (n .* t)) + sin((pi/3) .* t .* (n
.* t));
     elseif strcmp(Name,'QuadChirp'),
         sig = sin( (pi/3) .* t .* (n .* t.^2));
     elseif strcmp(Name,'MishMash'),  % QuadChirp + LinChirp +
HiSine
         sig = sin( (pi/3) .* t .* (n .* t.^2)) ;
         sig = sig +  sin( pi * (n * .6902) .* t);
         sig = sig +  sin(pi .* t .* (n .* .125 .* t));
     elseif strcmp(Name,'WernerSorrows'),
         sig = sin( pi .* t .* (n/2 .* t.^2)) ;
         sig = sig +  sin( pi * (n * .6902) .* t);
         sig = sig +  sin(pi .* t .* (n .* t));
         pos = [ .1 .13 .15 .23 .25 .40 .44 .65  .76 .78 .81];
         hgt = [ 4  5   3   4  5  4.2 2.1 4.3  3.1 5.1 4.2];
         wth = [.005 .005 .006 .01 .01 .03 .01 .01  .005 .008
.005];
         for j =1:length(pos)
            sig = sig + hgt(j)./( 1 + abs((t -
pos(j))./wth(j))).^4;
         end
     elseif strcmp(Name,'Leopold'),
         sig = (t == floor(.37 * n)/n);  % Kronecker
     elseif strcmp(Name,'Riemann'),
          sqn = round(sqrt(n));
         sig = t .* 0;  % Riemann's Non-differentiable Function
          sig((1:sqn).^2) = 1. ./ (1:sqn);
          sig = real(ifft(sig));
     elseif strcmp(Name,'HypChirps'), % Hyperbolic Chirps of
Mallat's book
          alpha= 15*n*pi/1024;
          beta    = 5*n*pi/1024;
          t    = (1.001:1:n+.001)./n;
          f1      = zeros(1,n);
          f2      = zeros(1,n);
          f1   = sin(alpha./(.8-t)).*(0.1<t).*(t<0.68);
          f2   = sin(beta./(.8-t)).*(0.1<t).*(t<0.75);
          M    = round(0.65*n);
          P    = floor(M/4);
          enveloppe = ones(1,M); % the rising cutoff function
           enveloppe(1:P) = (1+sin(-pi/2+((1:P)-ones(1,P))./(P-
1)*pi))/2;
           enveloppe(M-P+1:M) = reverse(enveloppe(1:P));
          env  = zeros(1,n);
          env(ceil(n/10):M+ceil(n/10)-1) = enveloppe(1:M);
          sig     = (f1+f2).*env;
     elseif strcmp(Name,'LinChirps'), % Linear Chirps of
Mallat's book
          b    = 100*n*pi/1024;
          a    = 250*n*pi/1024;
          t    = (1:n)./n;
          A1   = sqrt((t-1/n).*(1-t));
          sig  = A1.*(cos((a*(t).^2)) + cos((b*t+a*(t).^2)));
     elseif strcmp(Name,'Chirps'), % Mixture of Chirps of
Mallat's book
          t    = (1:n)./n.*10.*pi;
          f1   = cos(t.^2*n/1024);
          a    = 30*n/1024;
          t    = (1:n)./n.*pi;
          f2   = cos(a.*(t.^3));
          f2   = reverse(f2);
          ix   = (-n:n)./n.*20;
          g    = exp(-ix.^2*4*n/1024);
          i1   = (n/2+1:n/2+n);
          i2   = (n/8+1:n/8+n);
          j    = (1:n)/n;
          f3   = g(i1).*cos(50.*pi.*j*n/1024);
          f4   = g(i2).*cos(350.*pi.*j*n/1024);
          sig  = f1+f2+f3+f4;
      enveloppe = ones(1,n); % the rising cutoff function
      enveloppe(1:n/8) = (1+sin(-pi/2+((1:n/8)-
ones(1,n/8))./(n/8-1)*pi))/2;
      enveloppe(7*n/8+1:n) = reverse(enveloppe(1:n/8));
          sig  = sig.*enveloppe;
        elseif strcmp(Name,'Gabor'), % two modulated Gabor
functions in
                         % Mallat's book
          N = 512;
          t = (-N:N)*5/N;
          j = (1:N)./N;
          g = exp(-t.^2*20);
          i1 = (2*N/4+1:2*N/4+N);
          i2 = (N/4+1:N/4+N);
          sig1 = 3*g(i1).*exp(i*N/16.*pi.*j);
          sig2 = 3*g(i2).*exp(i*N/4.*pi.*j);
          sig = sig1+sig2;
     elseif strcmp(Name,'sineoneoverx'), % sin(1/x) in Mallat's
book
```

```
            N = 1024;
            i = (-N+1:N);
            i(N) = 1/100;
            i = i./(N-1);
            sig = sin(1.5./(i));
            sig = sig(513:1536);
      elseif strcmp(Name,'Cusp2'),
            N = 64;
            i = (1:N)./N;
            x = (1-sqrt(i)) + i/2 -.5;
            M = 8*N;
            sig = zeros(1,M);
            sig(M-1.5.*N+1:M-.5*N) = x;
            sig(M-2.5*N+2:M-1.5.*N+1) = reverse(x);
            sig(3*N+1:3*N + N) = .5*ones(1,N);
      elseif strcmp(Name,'SmoothCusp'),
            sig = MakeSignal('Cusp2');
            N = 64;
            M = 8*N;
            t = (1:M)/M;
            sigma = 0.01;
            g = exp(-.5.*(abs(t-
.5)./sigma).^2)./sigma./sqrt(2*pi);
            g = fftshift(g);
            sig2 = iconv(g',sig)'/M;
      elseif strcmp(Name,'Piece-Regular'),
            sig1=-15*MakeSignal('Bumps',n);
            t = (1:fix(n/12)) ./fix(n/12);
            sig2=-exp(4*t);
            t = (1:fix(n/7)) ./fix(n/7);
            sig5=exp(4*t)-exp(4);
            t = (1:fix(n/3)) ./fix(n/3);
            sigma=6/40;
            sig6=-70*exp(-((t-1/2).*(t-1/2))/(2*sigma^2));
            sig(1:fix(n/7))= sig6(1:fix(n/7));

      sig((fix(n/7)+1):fix(n/5))=0.5*sig6((fix(n/7)+1):fix(n/5));

      sig((fix(n/5)+1):fix(n/3))=sig6((fix(n/5)+1):fix(n/3));

      sig((fix(n/3)+1):fix(n/2))=sig1((fix(n/3)+1):fix(n/2));
            sig((fix(n/2)+1):(fix(n/2)+fix(n/12)))=sig2;
            sig((fix(n/2)+2*fix(n/12)):-
1:(fix(n/2)+fix(n/12)+1))=sig2;
sig(fix(n/2)+2*fix(n/12)+fix(n/20)+1:(fix(n/2)+2*fix(n/12)+3*fix(
n/20)))=...
-ones(1,fix(n/2)+2*fix(n/12)+3*fix(n/20)-fix(n/2)-2*fix(n/12)-
fix(n/20))*25;
            k=fix(n/2)+2*fix(n/12)+3*fix(n/20);
            sig((k+1):(k+fix(n/7)))=sig5;
            diff=n-5*fix(n/5);
            sig(5*fix(n/5)+1:n)=sig(diff:-1:1);
```

```
          % zero-mean
          bias=sum(sig)/n;
          sig=bias-sig;
     elseif strcmp(Name,'Piece-Polynomial'),
          t = (1:fix(n/5)) ./fix(n/5);
          sig1=20*(t.^3+t.^2+4);
          sig3=40*(2.*t.^3+t) + 100;
          sig2=10.*t.^3 + 45;
          sig4=16*t.^2+8.*t+16;
          sig5=20*(t+4);
          sig6(1:fix(n/10))=ones(1,fix(n/10));
          sig6=sig6*20;
          sig(1:fix(n/5))=sig1;
          sig(2*fix(n/5):-1:(fix(n/5)+1))=sig2;
          sig((2*fix(n/5)+1):3*fix(n/5))=sig3;
          sig((3*fix(n/5)+1):4*fix(n/5))=sig4;
          sig((4*fix(n/5)+1):5*fix(n/5))=sig5(fix(n/5):-1:1);
          diff=n-5*fix(n/5);
          sig(5*fix(n/5)+1:n)=sig(diff:-1:1);
          %sig((fix(n/20)+1):(fix(n/20)+fix(n/10)))=-
ones(1,fix(n/10))*20;

     sig((fix(n/20)+1):(fix(n/20)+fix(n/10)))=ones(1,fix(n/10))*
10;
          sig((n-fix(n/10)+1):(n+fix(n/20)-
fix(n/10)))=ones(1,fix(n/20))*150;
          % zero-mean
          bias=sum(sig)/n;
          sig=sig-bias;
     elseif strcmp(Name,'Gaussian'),
          sig=GWN(n,beta);
          g=zeros(1,n);
          lim=alpha*n;
          mult=pi/(2*alpha*n);
          g(1:lim)=(cos(mult*(1:lim))).^2;
          g((n/2+1):n)=g((n/2):-1:1);
          g = rnshift(g,n/2);
          g=g/norm(g);
          sig=iconv(g,sig);
       else
         disp(sprintf('MakeSignal: I don*t recognize
<<%s>>',Name))
         disp('Allowable Names are:')
            disp('HeaviSine'),
            disp('Bumps'),
            disp('Blocks'),
            disp('Doppler'),
            disp('Ramp'),
            disp('Cusp'),
            disp('Crease'),
            disp('Sing'),
            disp('HiSine'),
            disp('LoSine'),
            disp('LinChirp'),
            disp('TwoChirp'),
            disp('QuadChirp'),
            disp('MishMash'),
            disp('WernerSorrows'),
            disp('Leopold'),
            disp('Sing'),
            disp('HiSine'),
            disp('LoSine'),
            disp('LinChirp'),
            disp('TwoChirp'),
            disp('QuadChirp'),
            disp('MishMash'),
            disp('WernerSorrows'),
            disp('Leopold'),
            disp('Riemann'),
            disp('HypChirps'),
            disp('LinChirps'),
            disp('Chirps'),
            disp('sineoneoverx'),
            disp('Cusp2'),
            disp('SmoothCusp'),
            disp('Gabor'),
            disp('Piece-Regular');
            disp('Piece-Polynomial');
            disp('Gaussian');
     end

%
% Originally made by David L. Donoho.
% Function has been enhanced.


%
% Part of WaveLab Version 802
% Built Sunday, October 3, 1999 8:52:27 AM
% This is Copyrighted Material
% For Copying permissions see COPYING.m
% Comments? e-mail wavelab@stat.stanford.edu
%
```

2. Kurtosistest

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          %
% kurtosistest: implements thebootstrap-based hypothesis   %
%               test H: kurtosis of the input=3 against    %
%               kurtosis of input~=3 and performs soft and %
%               hard thresholding on the data              %
%                                                          %
% SYNTAX    : [soft,hard,kur] = kurtosistest[data,kurlim]  %
%                                                          %
% INPUT     : data: data sequence to be tested             %
%             kurlim: kurtosis limit not to be exceeded    %
%                     thresholding                         %
%                                                          %
% OUTPUT    : soft: soft thresholded data                  %
%             hard: hard thresholded data                  %
%             kur: kurtosis value used in soft thresholding%
%                                                          %
% SUB FUNC  : none                                         %
%                                                          %
% Written by Hasan E. KAN                                  %
%                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [varargout]=kurtosistest(varargin);

datain=varargin{1};
kurlim=varargin{2};

hardout=zeros(size(datain));

test=boottest(datain,'kurtosis',3,1,.05,99,25);
if test==1
    softout=datain;
    hardout=datain;
    kur=-1;
else
    storage=[];temp=datain;
    kur=mean(bootstrp(99,'kurtosis',datain));
    while kur>kurlim
        if length(storage)>round(length(temp)/3)
            kur=kurlim;
        else
            locations=find(abs(temp)==max(abs(temp)))';
            storage=[storage; locations];
            temp(locations)=0;
            kur=mean(bootstrp(99,'kurtosis',temp));
        end
    end
    softout=datain*abs(3-kur)/(kurlim-3);
    if ~isempty(storage)
        softout(storage)=datain(storage);
        hardout(storage)=datain(storage);
    end
end

varargout{1} = softout;
varargout{2} = hardout;
varargout{3} = kur;
```

APPENDIX B. SIMULATION RESULTS

Simulation results for the three test signal types are presented in detail in this Appendix. Descriptions for these test signals were given in Chapter 4. Results include performance criteria obtained for the MSE and distance measure defined in Chapter 4 for SNR levels between –6 to 6 dB, for one hundred trials per SNR level. Finally, the original, noisy and the recovered versions of one trial are shown for SNR values –6 dB, 0 dB and 6 dB to provide a visual representation of the FFT-based denoising scheme.

A. FFT-BASED DENOISING

1. Sinusoidal Signal

The frequency for the sinusoidal signal was selected randomly for each of the 100 trials so as to generalize the simulation results. The frequency was limited to avoid aliasing or DC signals. Minimum and maximum allowed frequencies for the sinusoidal test signal in the simulations were $2/512$ Hz and $241/512$ Hz respectively, where the sampling frequency was selected as 1 Hz. Figure 13 shows the performance curves for FFT-based denoising scheme. These curves indicate that the thresholding schemes perform similarly on this type of signal. Figures 14 to 16 illustrate noisy and recovered versions of a sinusoidal signal at SNR values of –6 dB, 0 dB and 6 dB respectively.

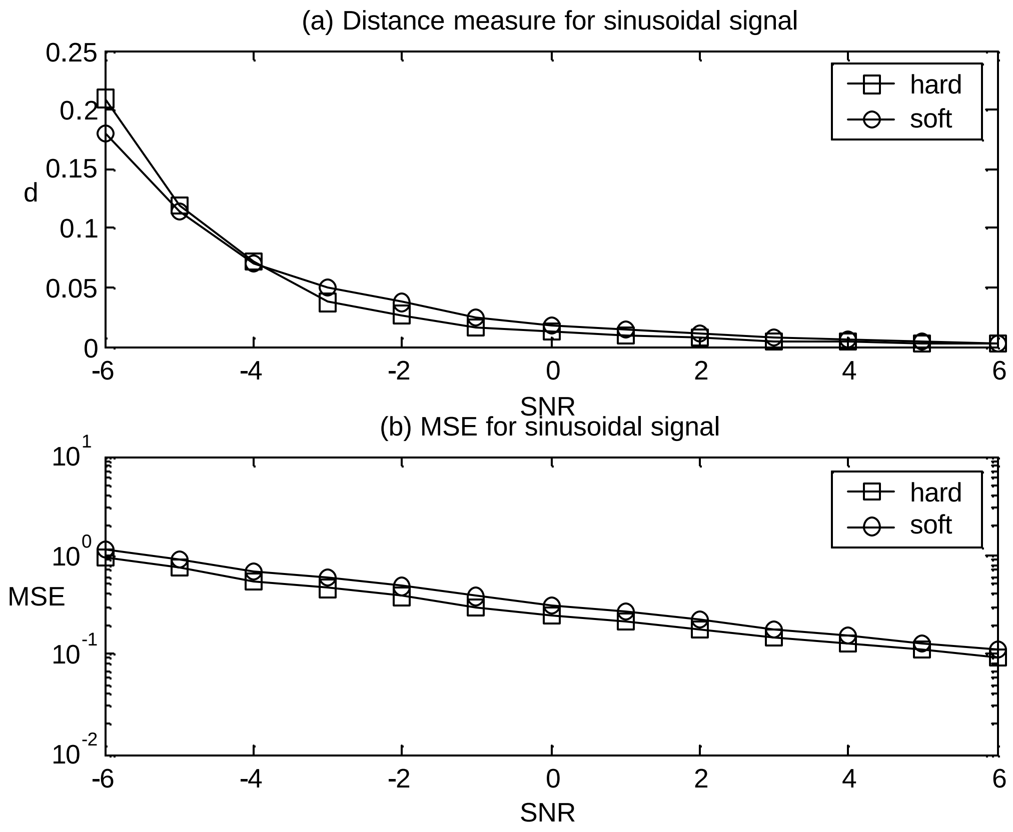

Figure 13. FFT-based denoising for sinusoidal signal type.

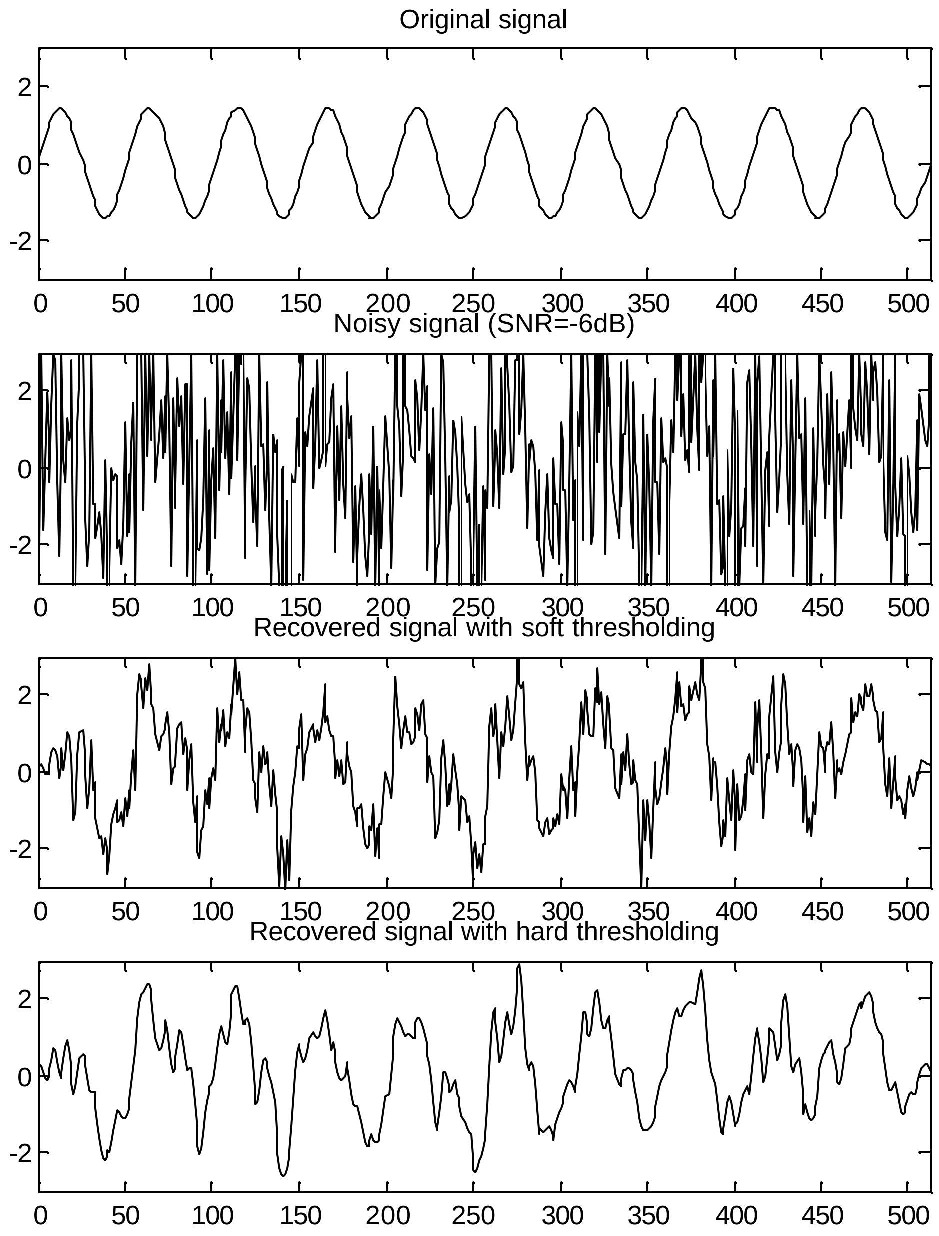

Figure 14. FFT-based denoising for sinusoidal signal type, SNR=–6 dB.

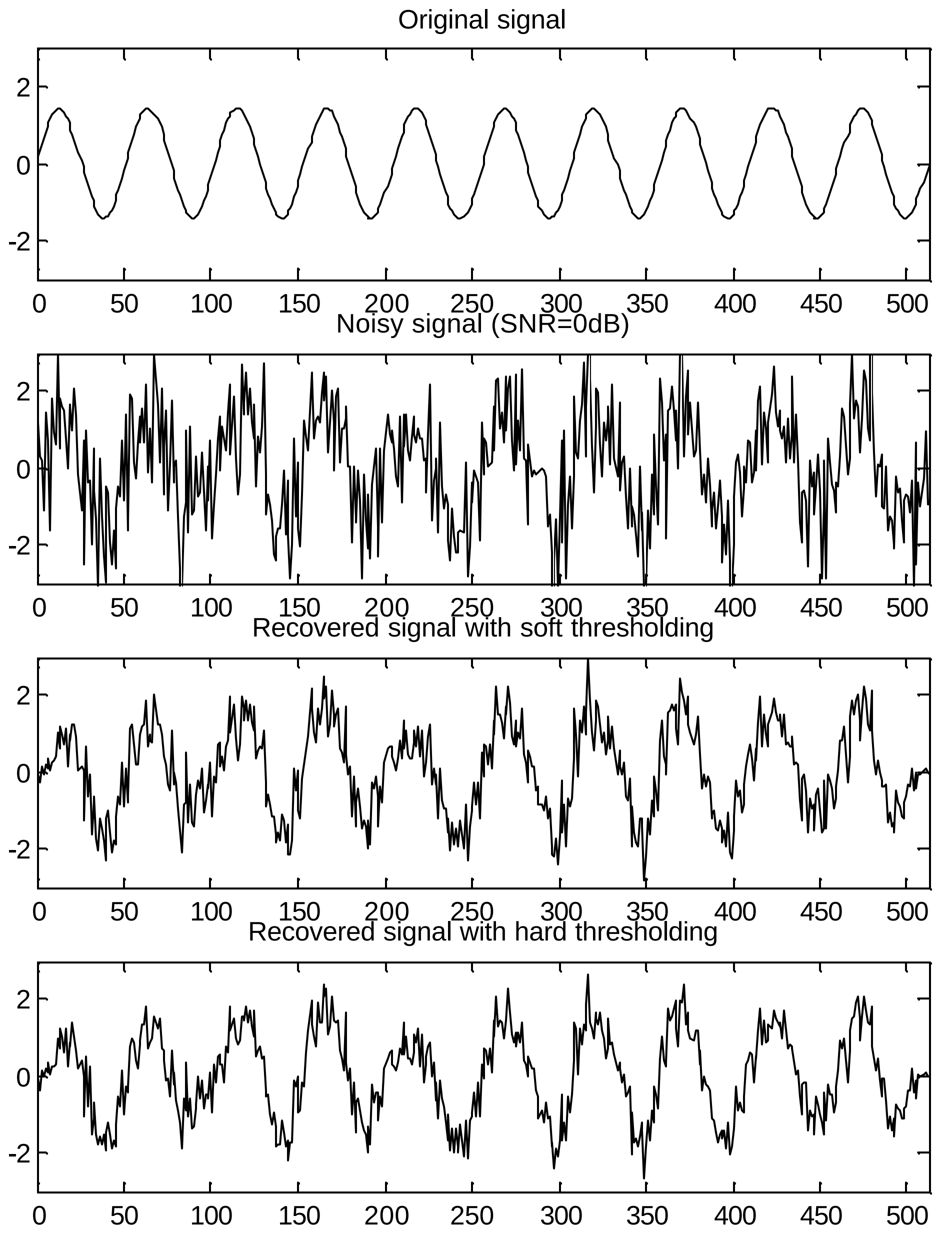

Figure 15. FFT-based denoising for sinusoidal signal type, SNR=0 dB.

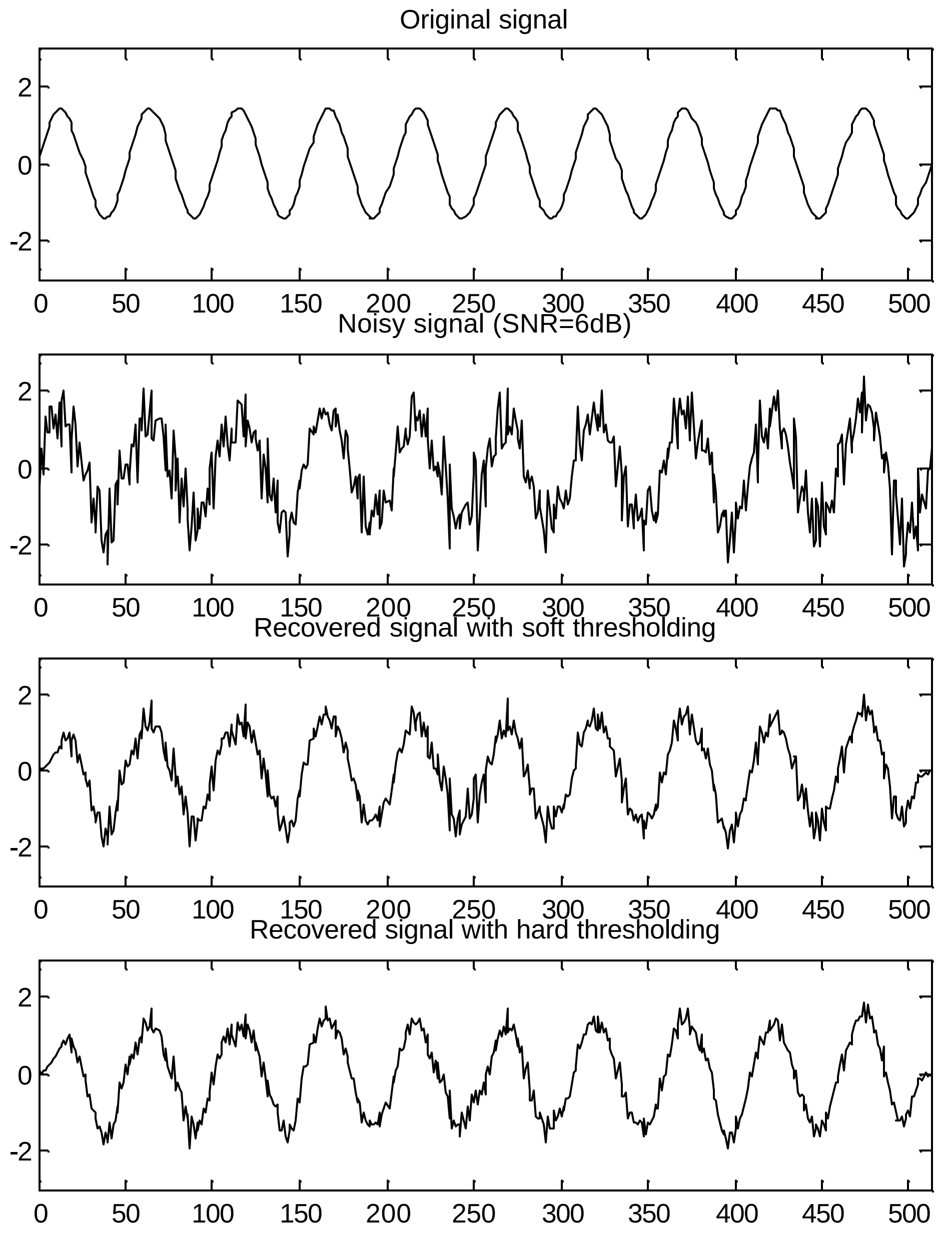

Figure 16. FFT-based denoising for sinusoidal signal type, SNR=6 dB.

2. Constant Amplitude Chirp Signal

Soft and hard thresholding perform similarly according to MSE. However, the distance measure shows that soft thresholding outperforms hard thresholding, especially at low and medium SNRs. This result can be seen in Figure 17. Figures 18 to 20 illustrate noisy and recovered versions of a sinusoidal signal at SNR values of –6 dB, 0 dB and 6 dB respectively.

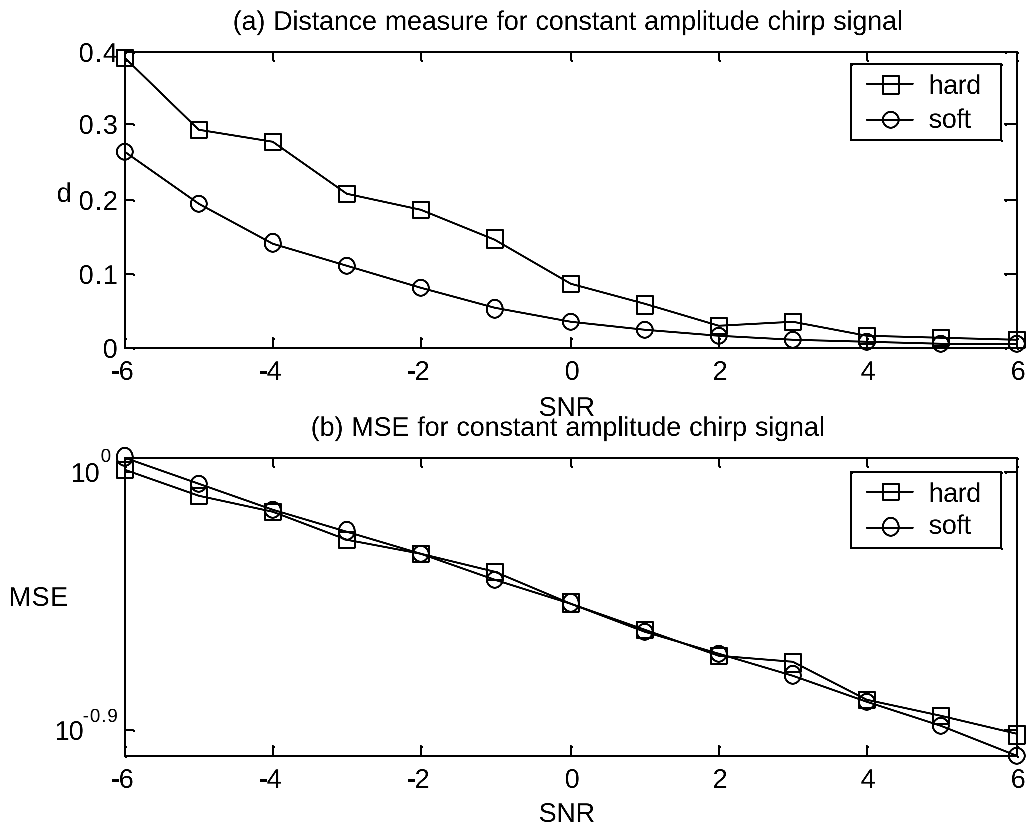

Figure 17. FFT-based denoising for constant amplitude chirp type signal.

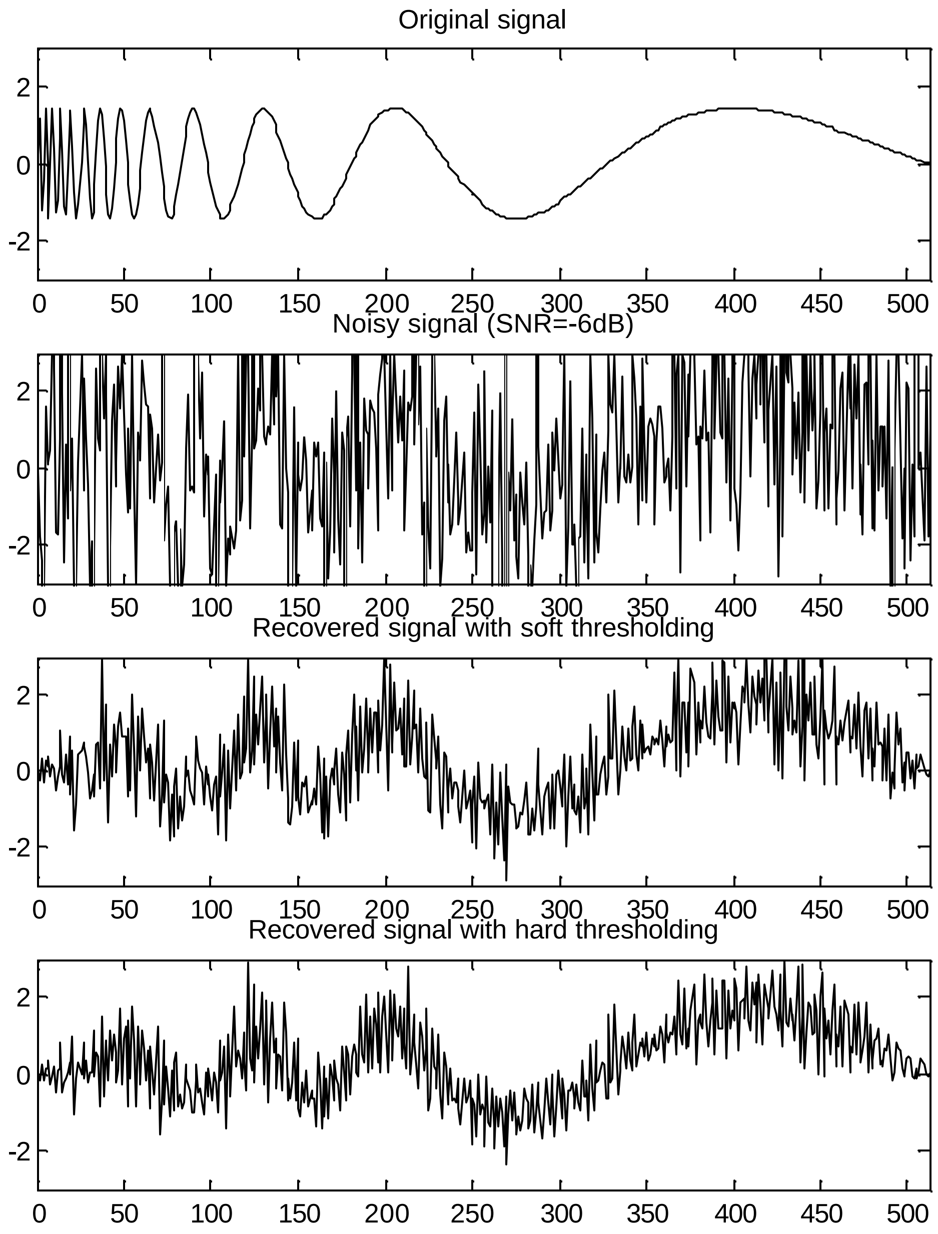

Figure 18. FFT-based denoising for constant amplitude chirp type, SNR=–6 dB.

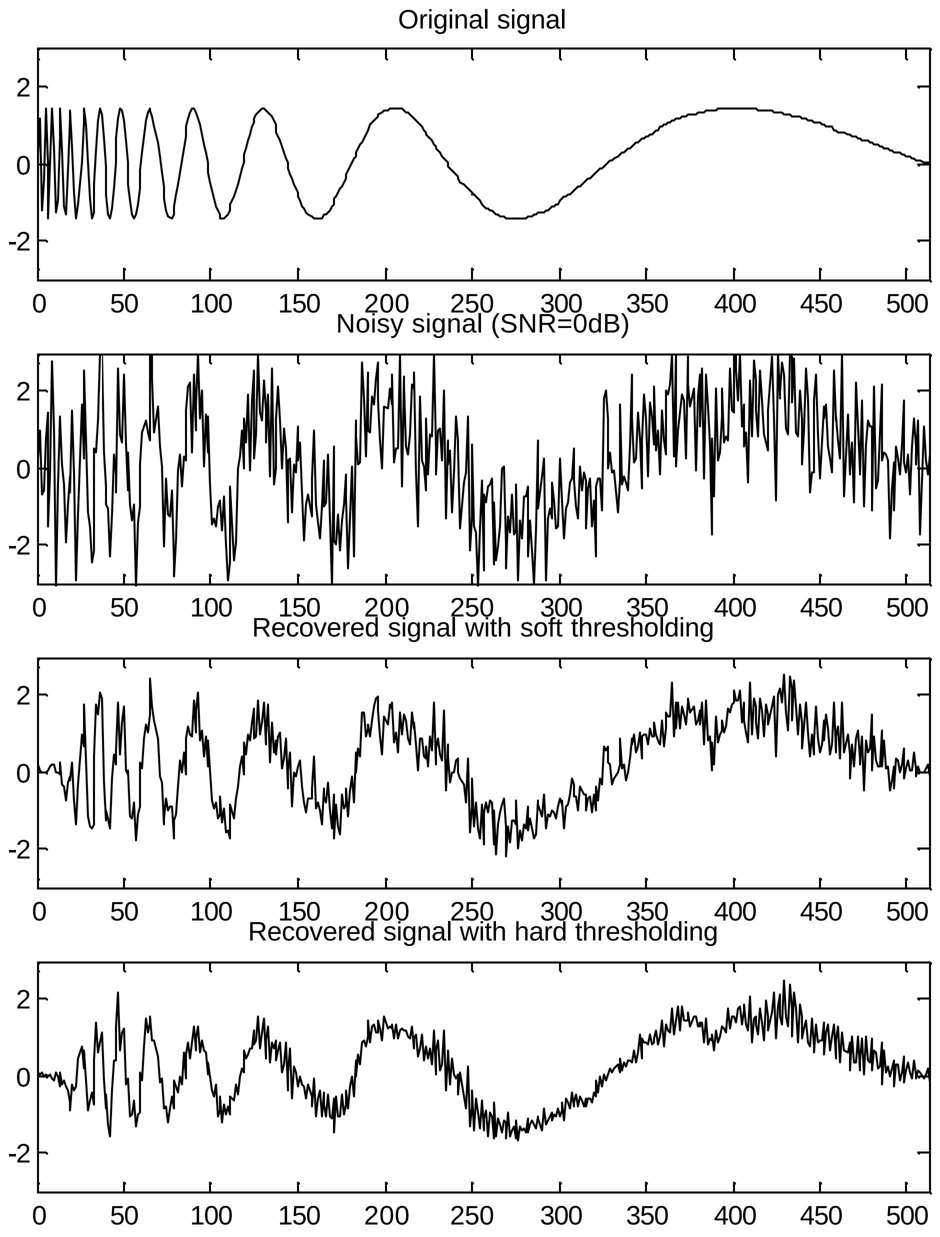

Figure 19. FFT-based denoising for constant amplitude chirp type, SNR=0 dB.

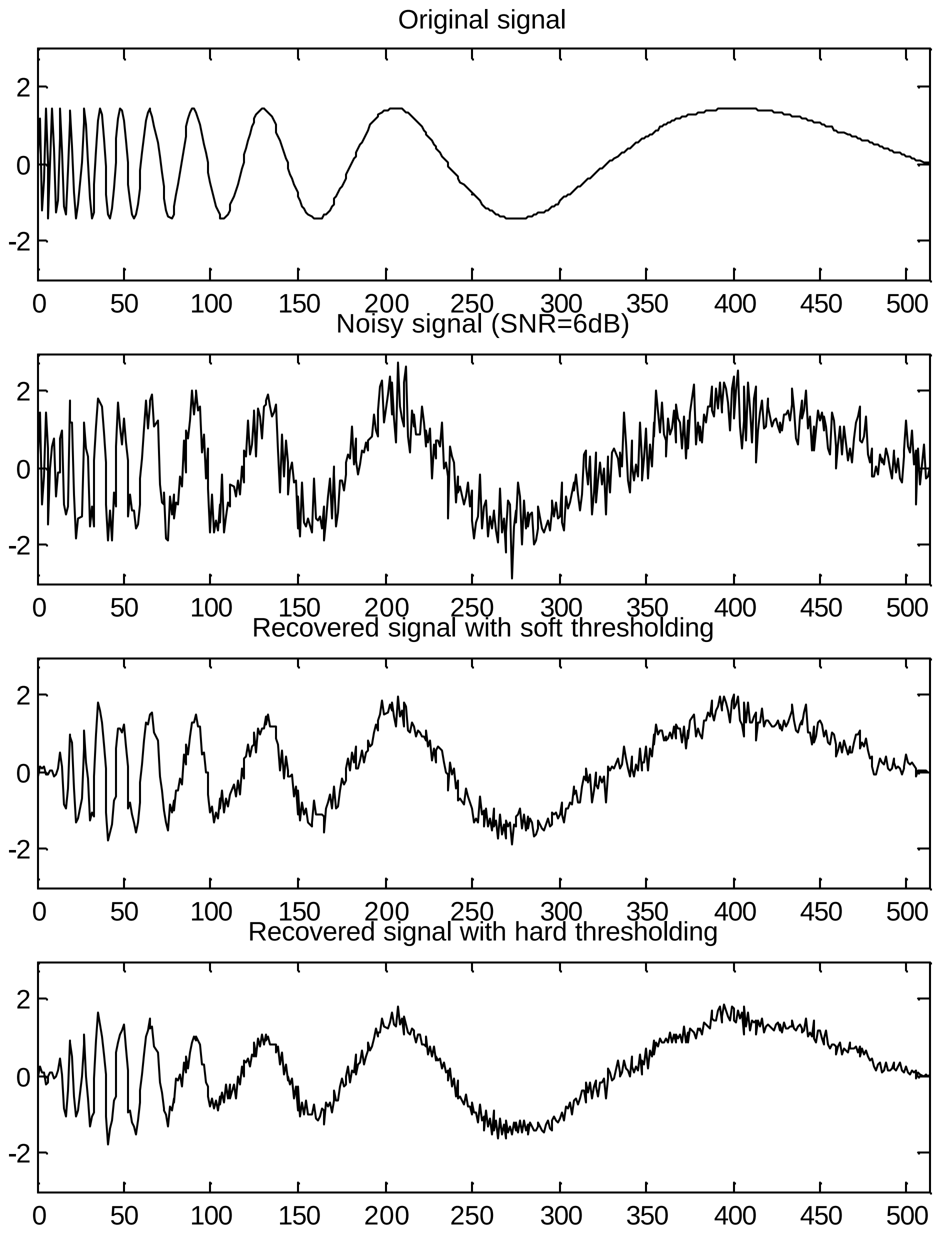

Figure 20. FFT-based denoising for constant amplitude chirp type, SNR=6 dB.

3. Chirp with an Amplitude that Increases in an RC Time Constant Fashion

The results shown in Figure 21 indicate that the thresholding schemes have similar MSE performances at SNR values lower than –3 dB, whereas soft thresholding performs better at higher SNRs. According to the distance measure *d*, soft thresholding is better at all SNR levels. Figures 22 to 24 illustrate noisy and recovered versions of a sinusoidal signal at SNR values of –6 dB, 0 dB and 6 dB respectively.

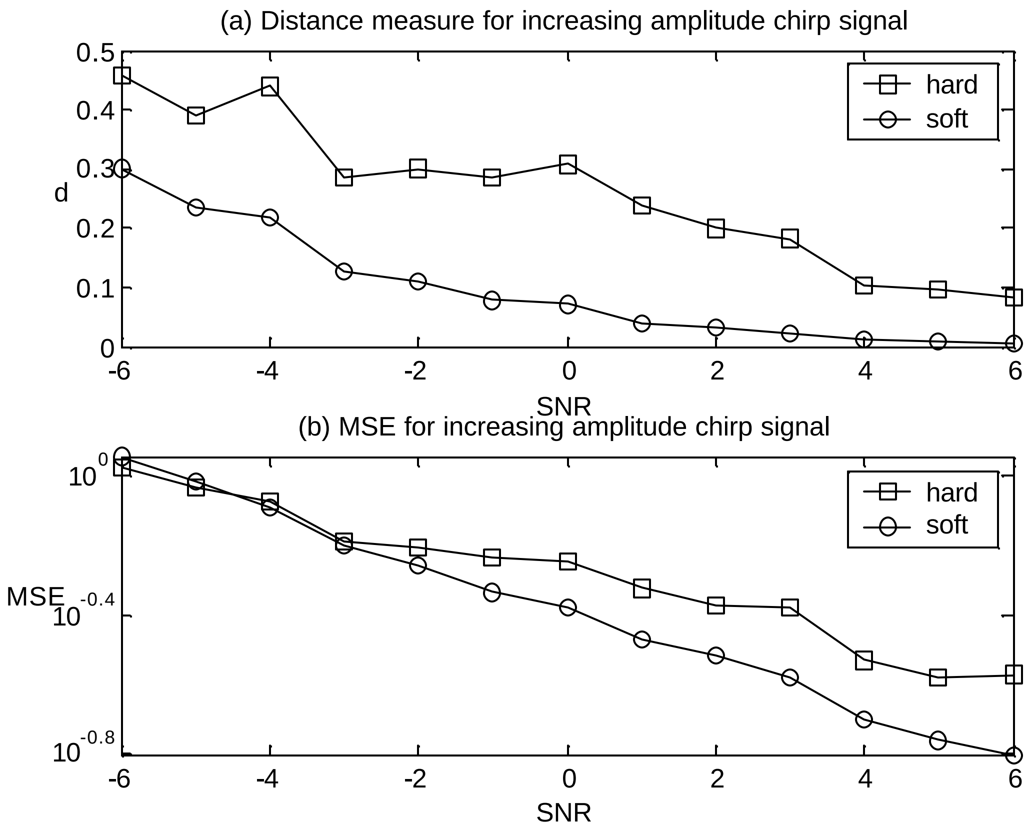

Figure 21. FFT-based denoising for increasing amplitude chirp type signal.

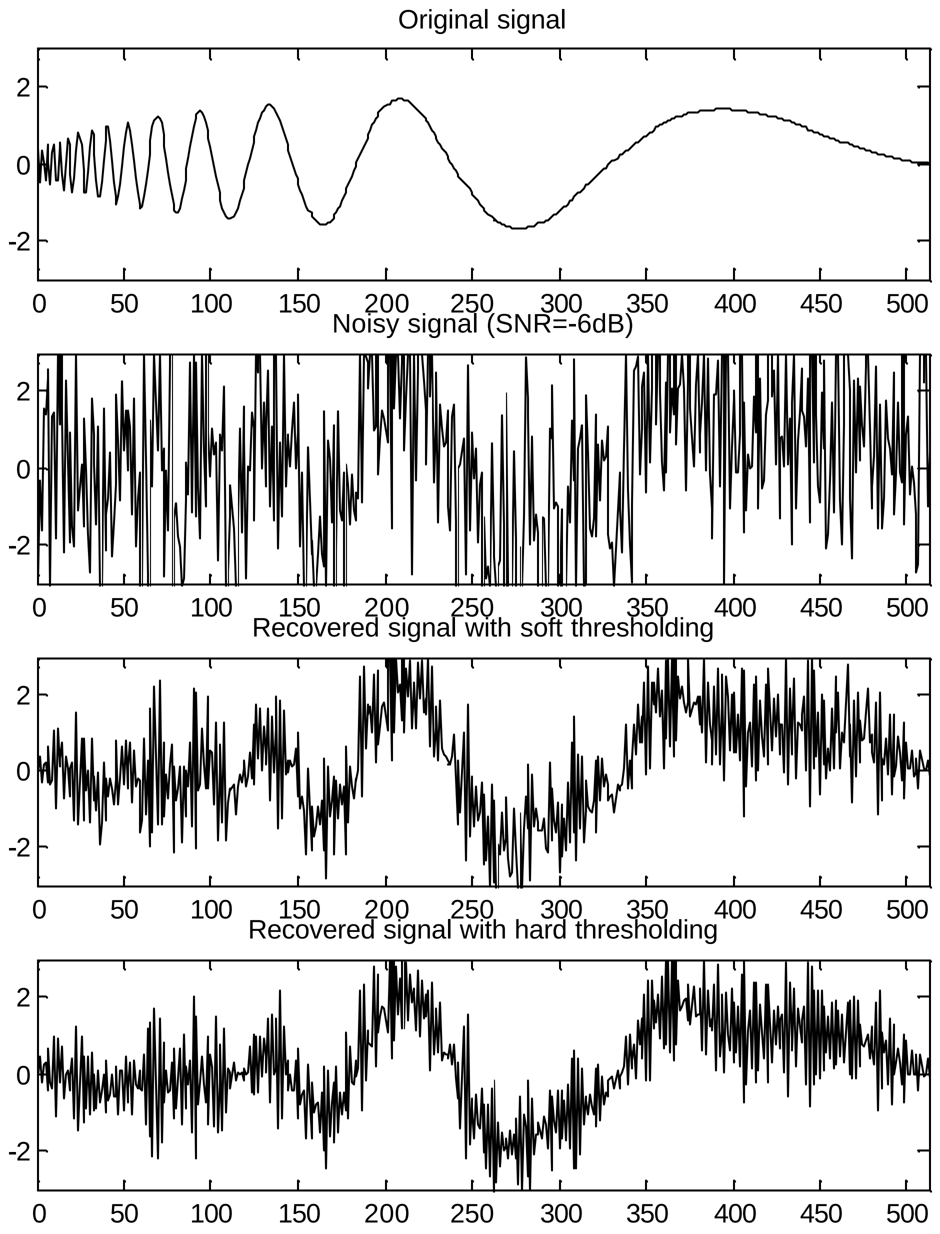

Figure 22. FFT-based denoising for increasing amplitude chirp type, SNR=–6 dB.

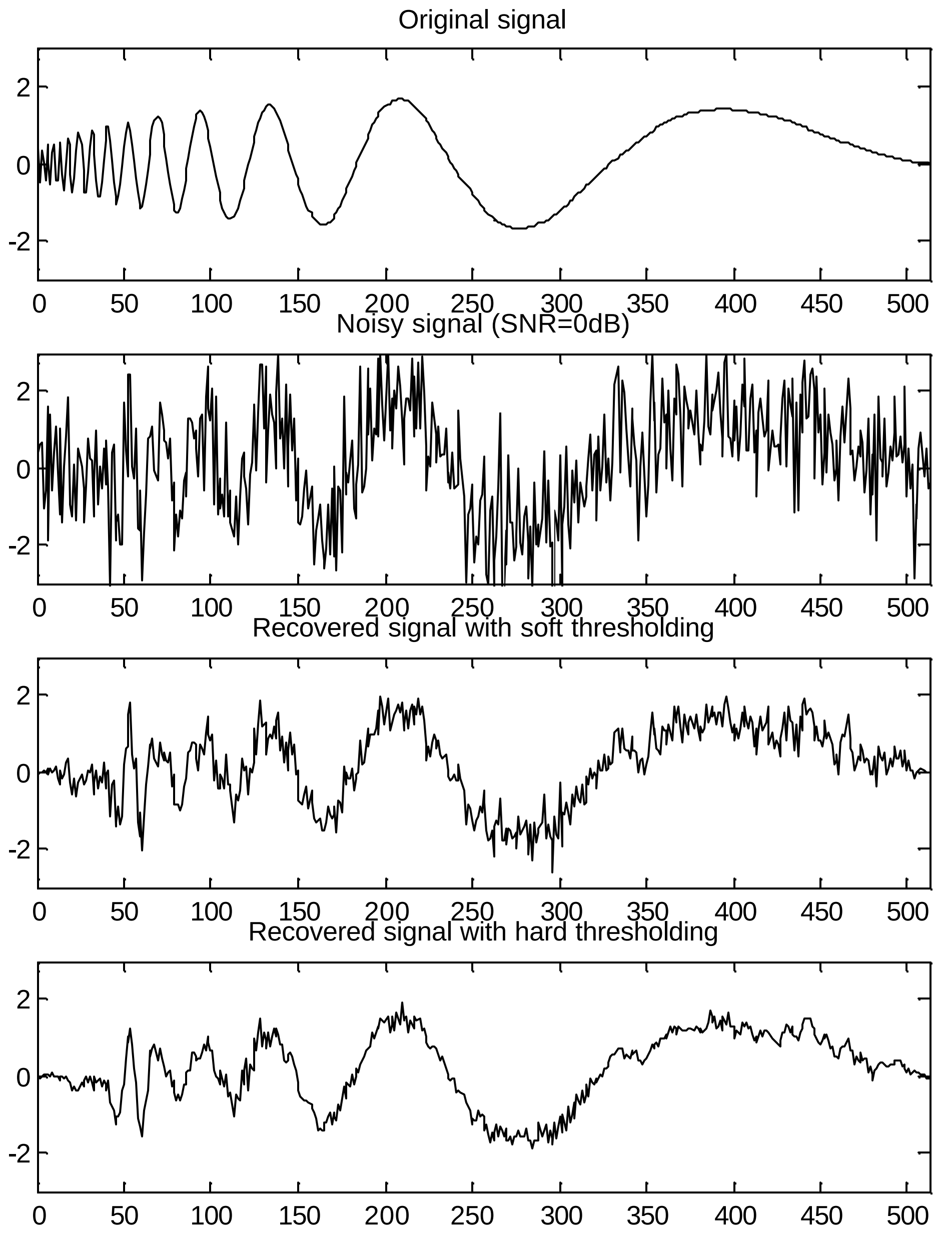

Figure 23. FFT-based denoising for increasing amplitude chirp type, SNR=0 dB.

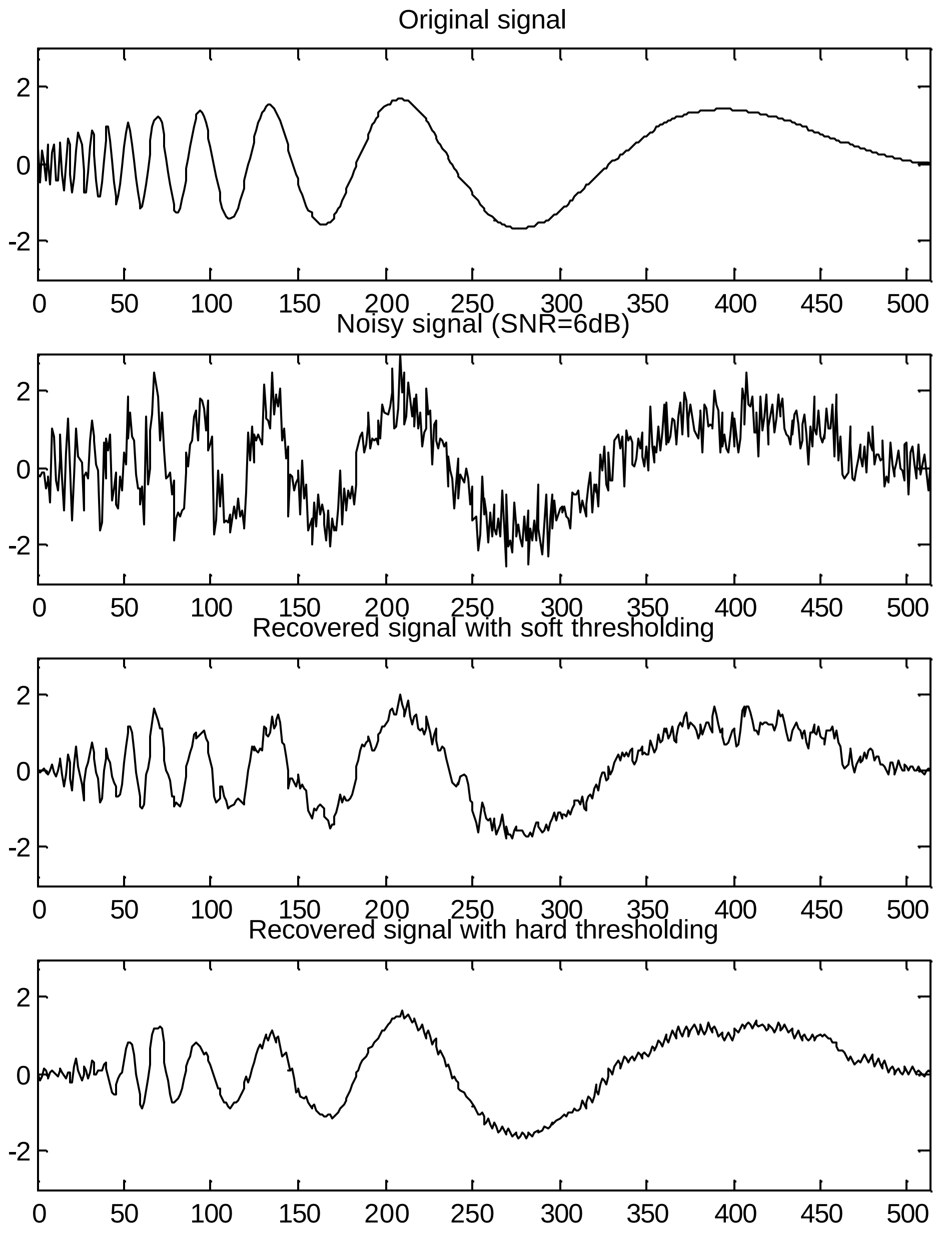

Figure 24. FFT-based denoising for increasing amplitude chirp type, SNR=6 dB.

B. WT-BASED DENOISING

Many different parameters affect the performance of the WT-based denoising scheme such as the number of decomposition levels, processing method for the approximation coefficients and the order of the Daubechies wavelet used. Results show no significant difference among the Daubechies wavelet orders considered. Only best results for each signal are illustrated in this section. Results shown include MSE and distance measure performances for SNR levels between –6 to 6 dB, where one hundred trials are used for each SNR level. Noisy and recovered versions of one trial are shown for SNR values –6 dB, 0 dB and 6 dB to provide a visual representation of the WT-based denoising scheme.

1. Sinusoidal Signal

Soft thresholding outperforms hard thresholding at low and medium SNR levels as shown in Figure 25. However, it does not perform as good as any of the thresholding methods used in the FFT-based denoising case for this signal type. Noisy and recovered versions of a sinusoidal signal at SNR values of –6 dB, 0 dB and 6 dB are shown in Figures 26 to 28.

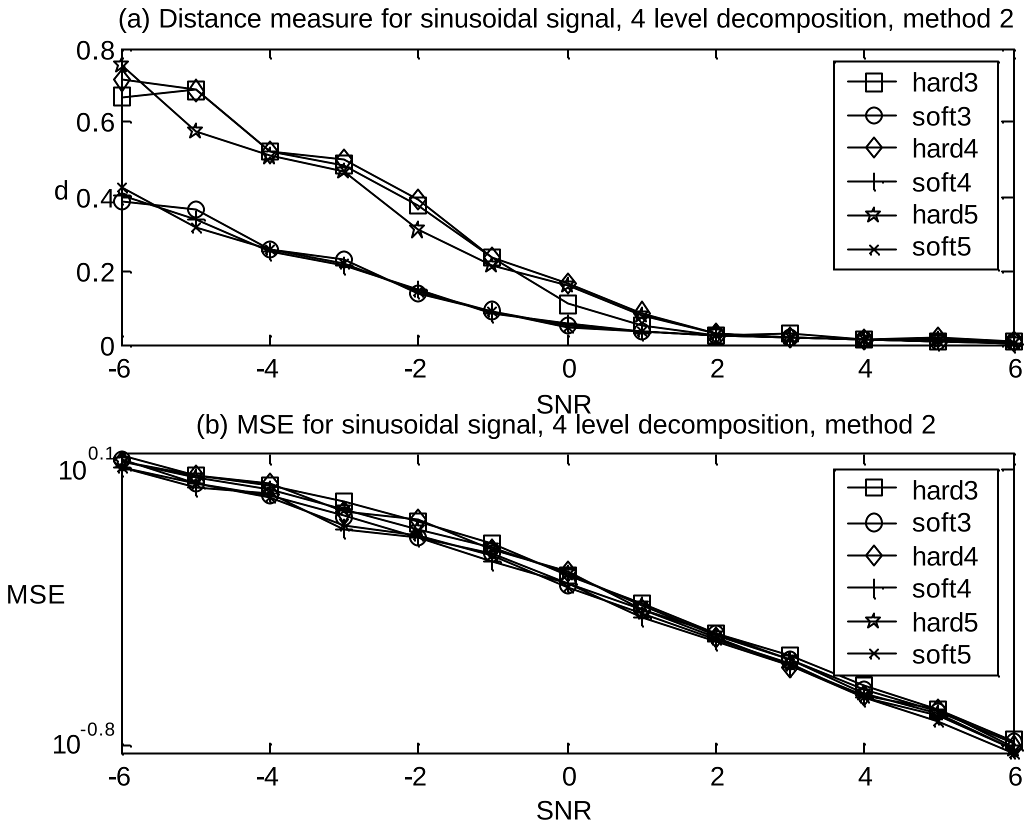

Figure 25. WT-based denoising for sinusoidal signal type, 4 level, method 2, with Daubechies wavelet orders 3, 4 and 5.

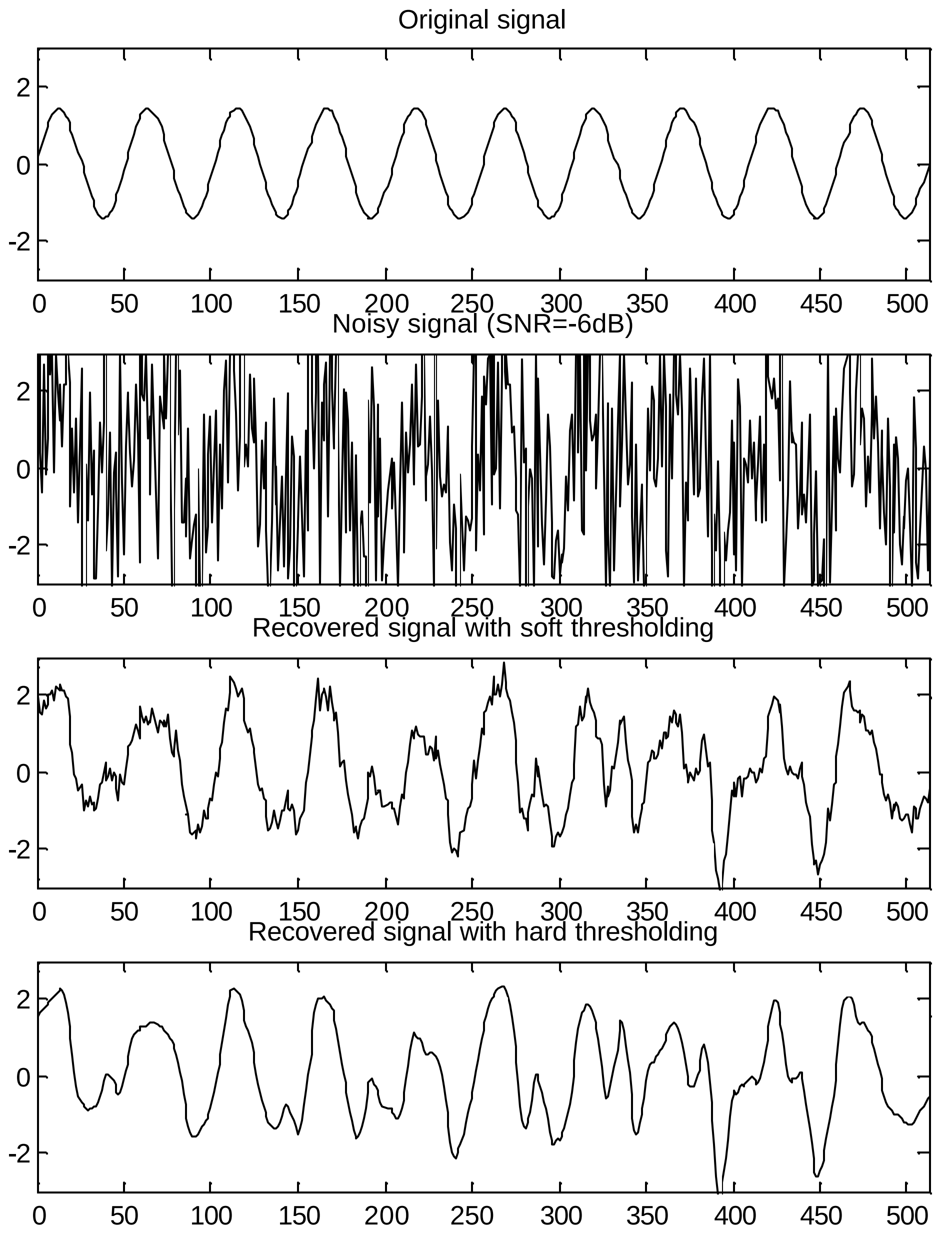

Figure 26. WT-based denoising for sinusoidal signal type, 4 level, method 2, Daubechies wavelet 4, SNR=–6 dB.

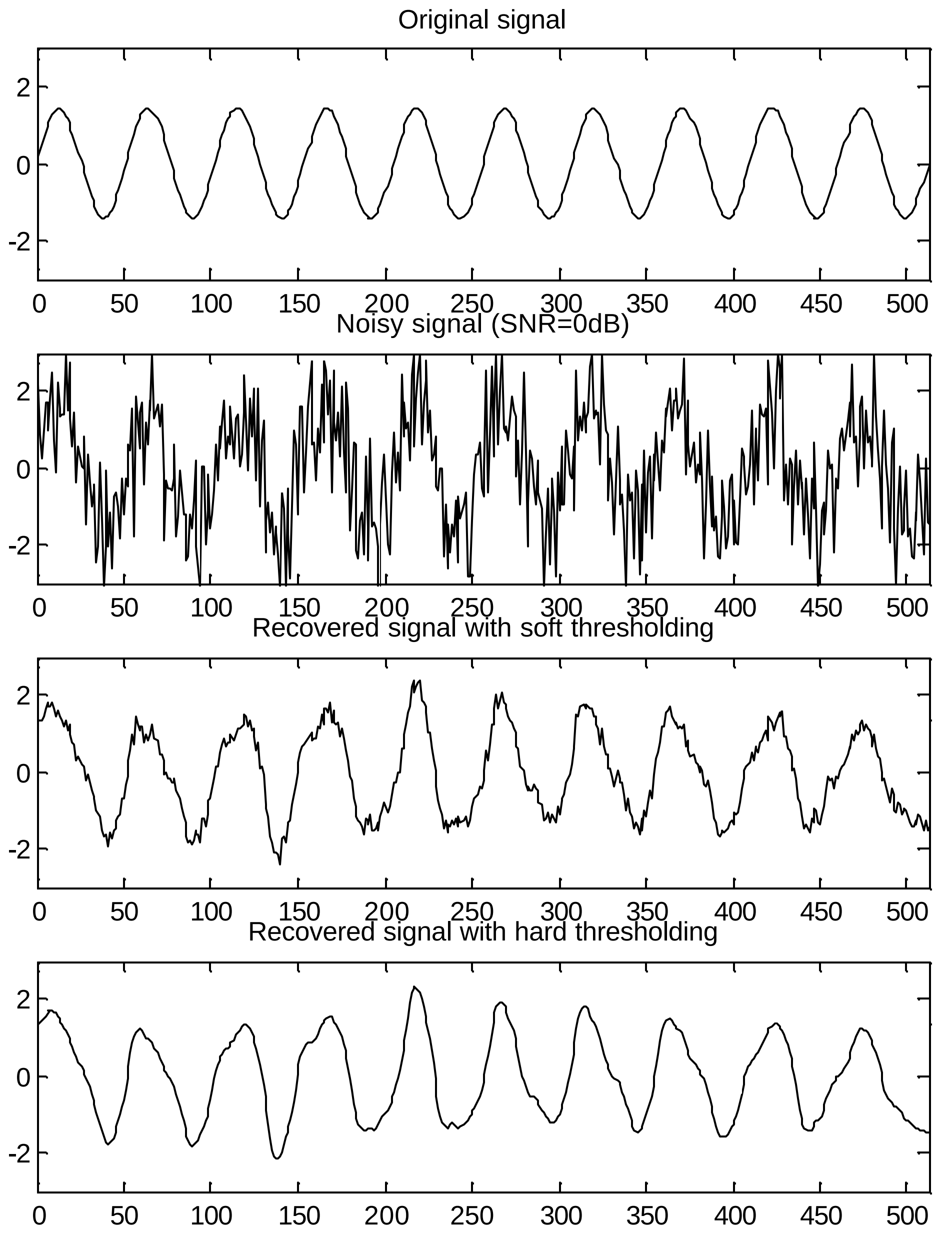

Figure 27. WT-based denoising for sinusoidal signal type, 4 level, method 2, Daubechies wavelet 4, SNR=0 dB.

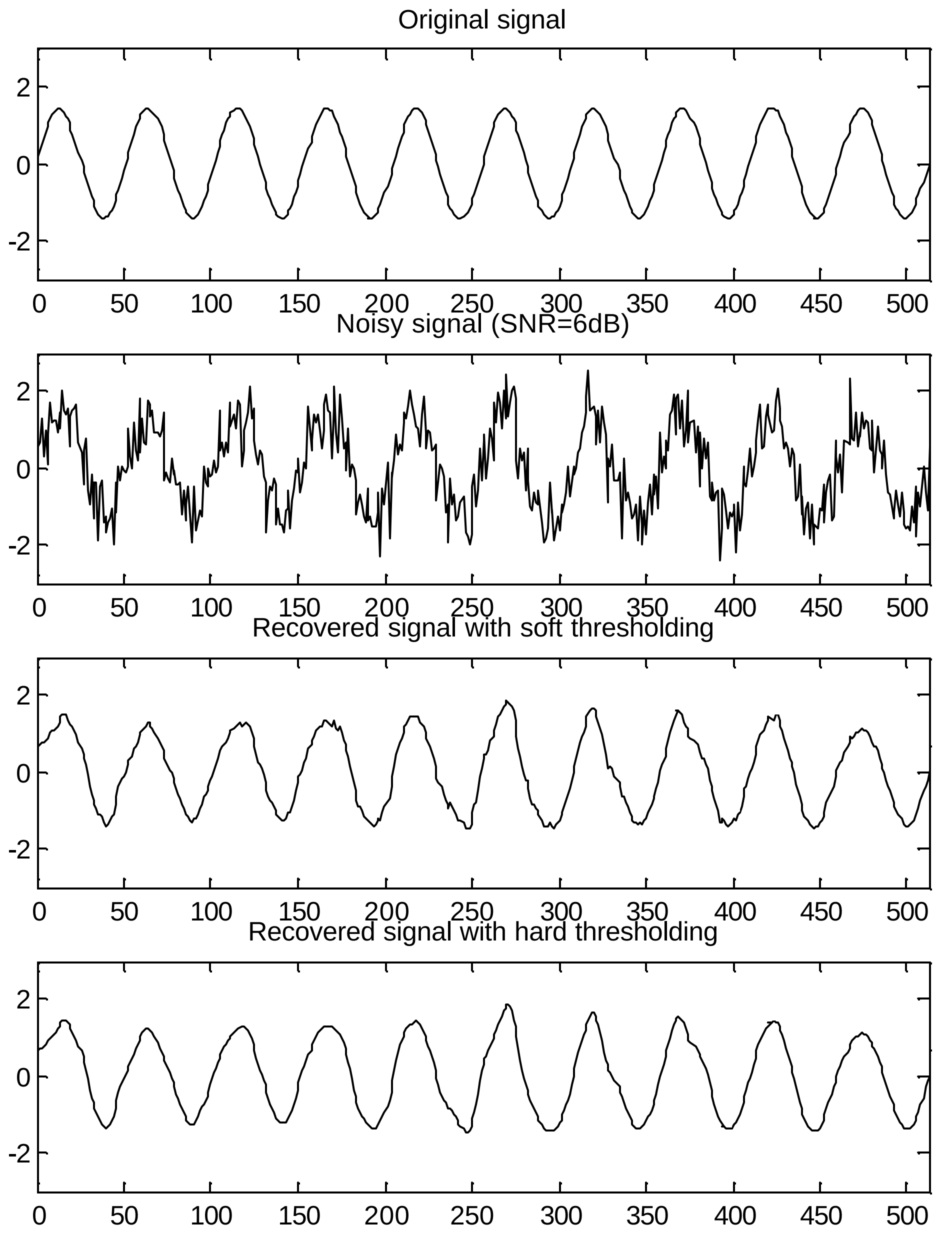

Figure 28. WT-based denoising for sinusoidal signal type, 4 level, method 2, Daubechies wavelet 4, SNR=6 dB.

2. Constant Amplitude Chirp

Results in Figure 29 show that soft and hard thresholding performances are similar for this signal when using WT-based denoising scheme. The WT-based scheme performs better than the FFT-based scheme on this signal. Figures 30 to 32 illustrate noisy and recovered versions of a sinusoidal signal at SNR values of –6 dB, 0 dB and 6 dB respectively.

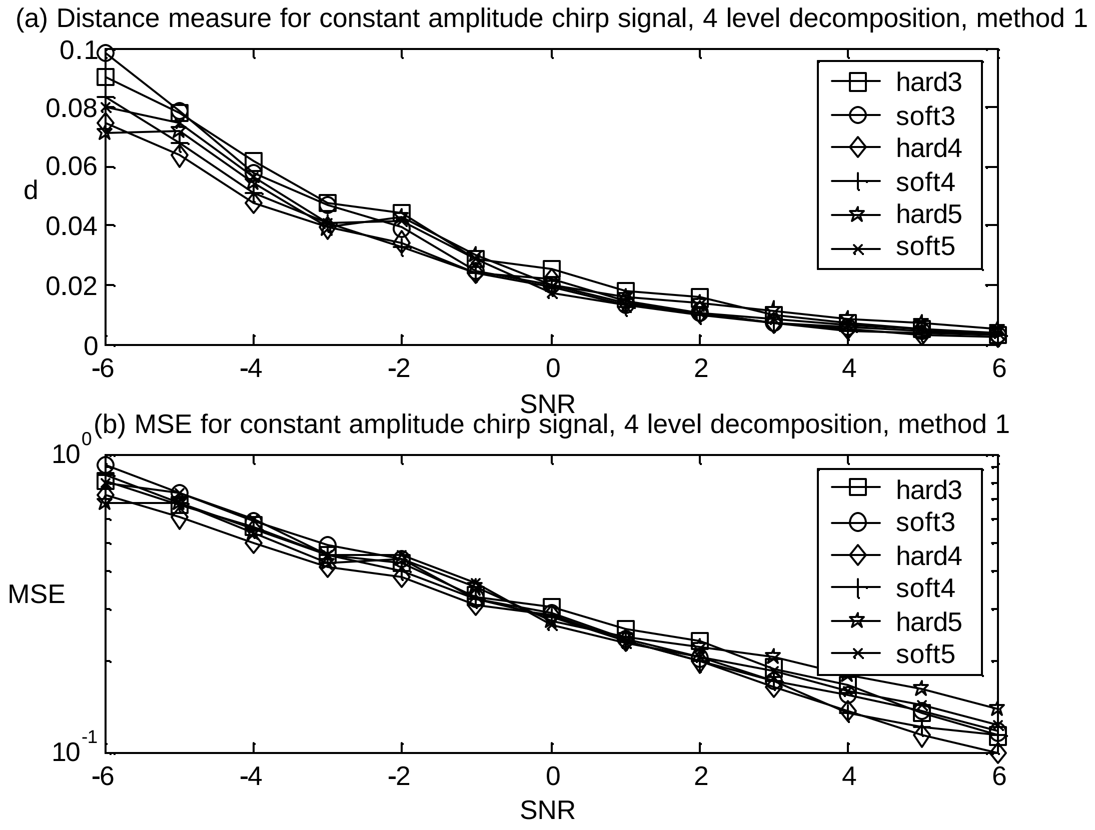

Figure 29. WT-based denoising for constant amplitude chirp type, 4 level, method 1, with Daubechies wavelet orders 3, 4 and 5.

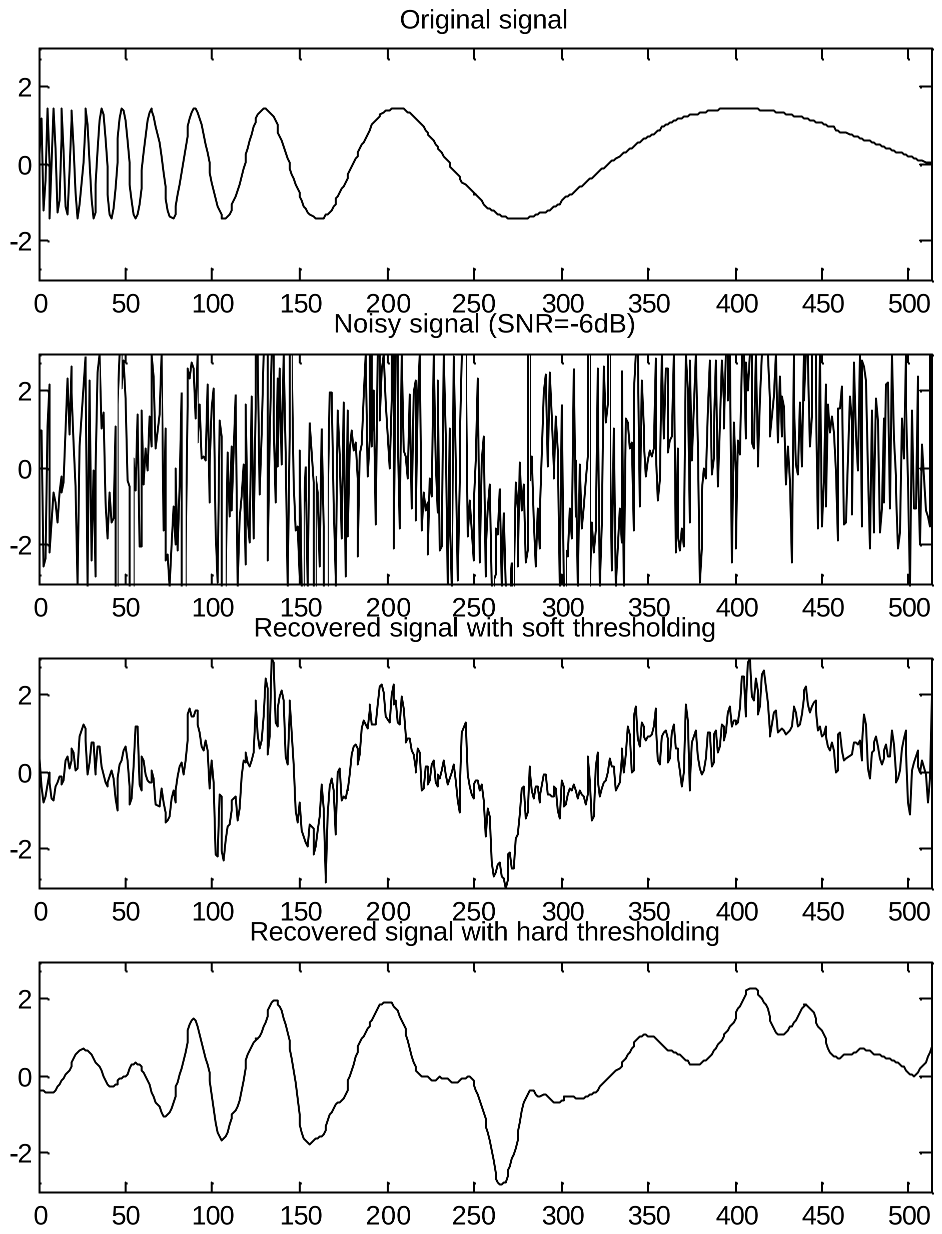

Figure 30. WT-based denoising for constant amplitude chirp type, 4 level, method 1, Daubechies wavelet 4, SNR=–6 dB.

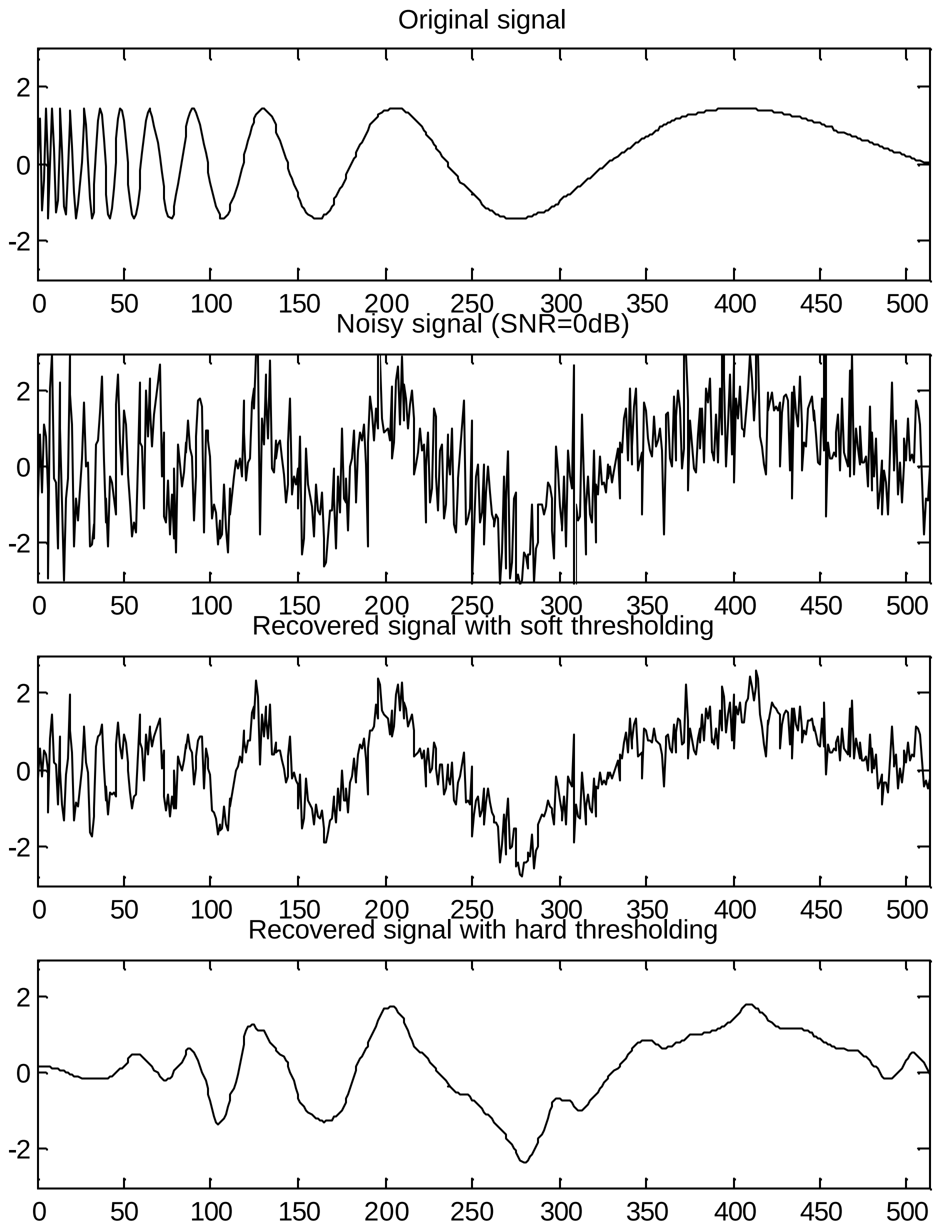

Figure 31. WT-based denoising for constant amplitude chirp type, 4 level, method 1, Daubechies wavelet 4, SNR=0 dB.

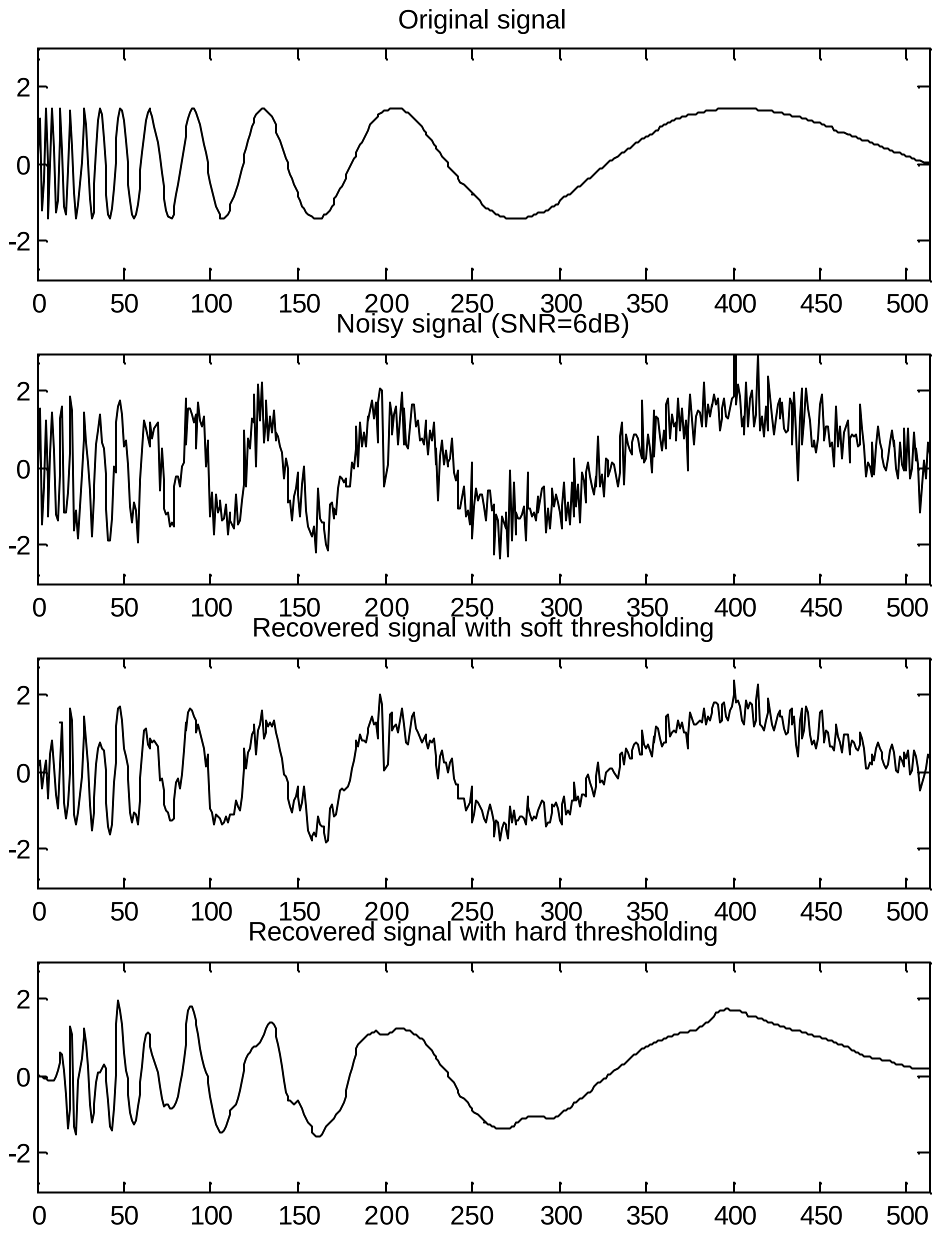

Figure 32. WT-based denoising for constant amplitude chirp type, 4 level, method 1, Daubechies wavelet 4, SNR=6 dB.

3. Chirp with an Amplitude that Increases in an RC Time Constant Fashion

Soft and hard thresholding methods perform similarly according to the performance criteria shown in Figure 33. WT-based denoising scheme outperforms FFT-based denoising scheme on this signal. Figures 34 to 36 illustrate noisy and recovered versions of a sinusoidal signal at SNR values of –6 dB, 0 dB and 6 dB respectively.

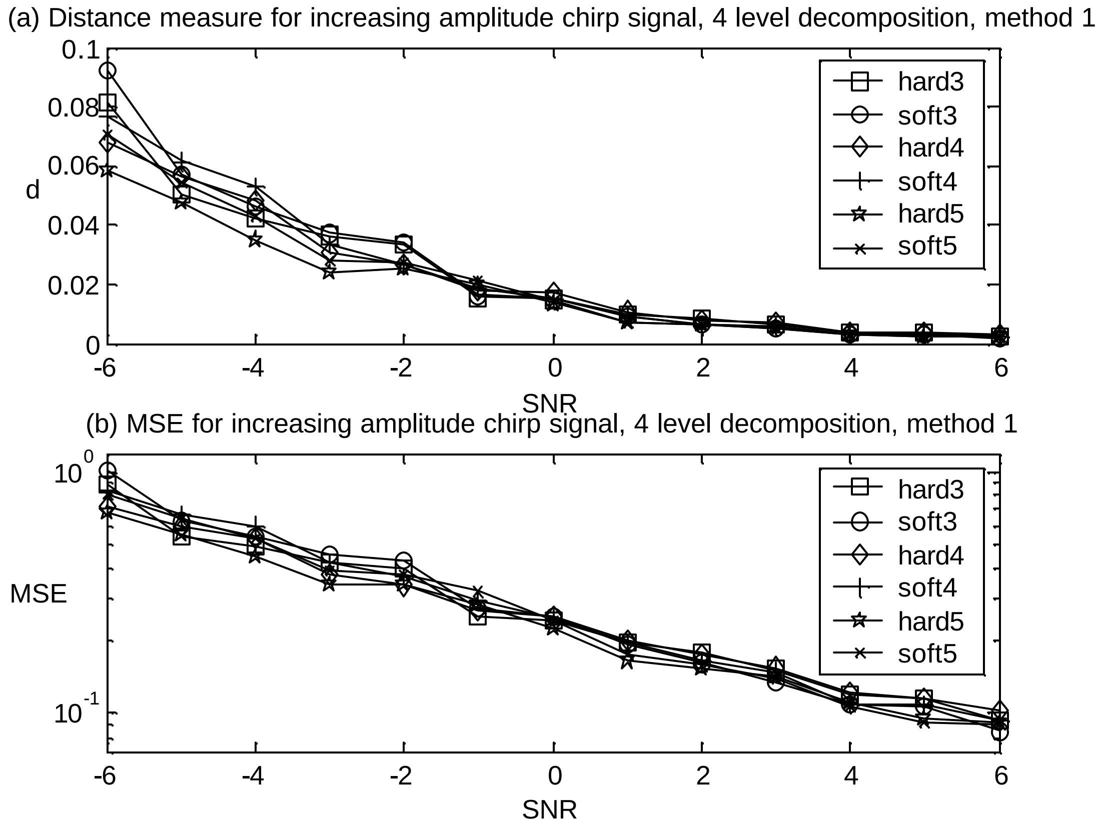

Figure 33. WT-based denoising for increasing amplitude chirp type, 4 level, method 1, with Daubechies wavelet orders 3, 4 and 5.

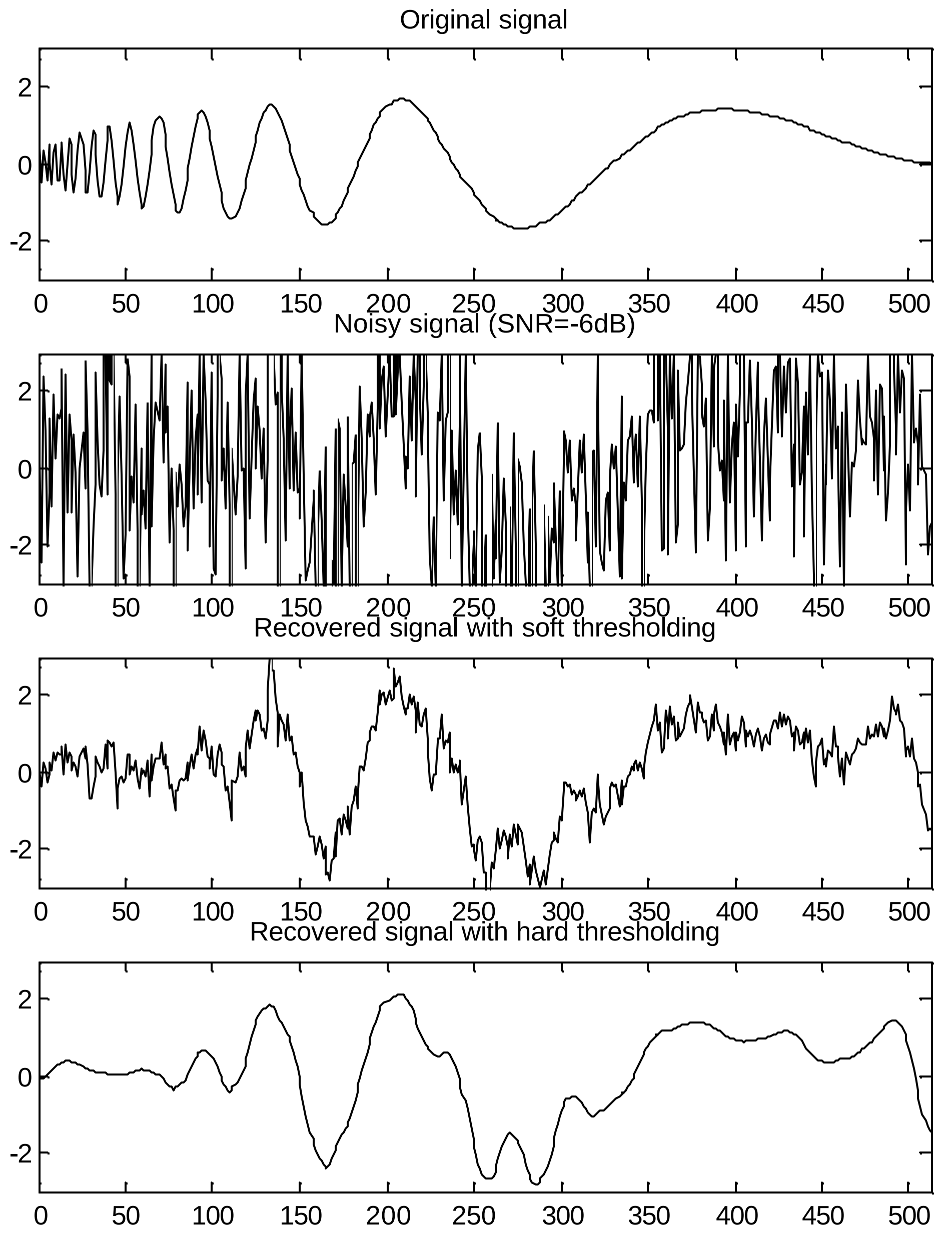

Figure 34. WT-based denoising for increasing amplitude chirp type, 4 level, method 1, Daubechies wavelet 5, SNR=–6 dB.

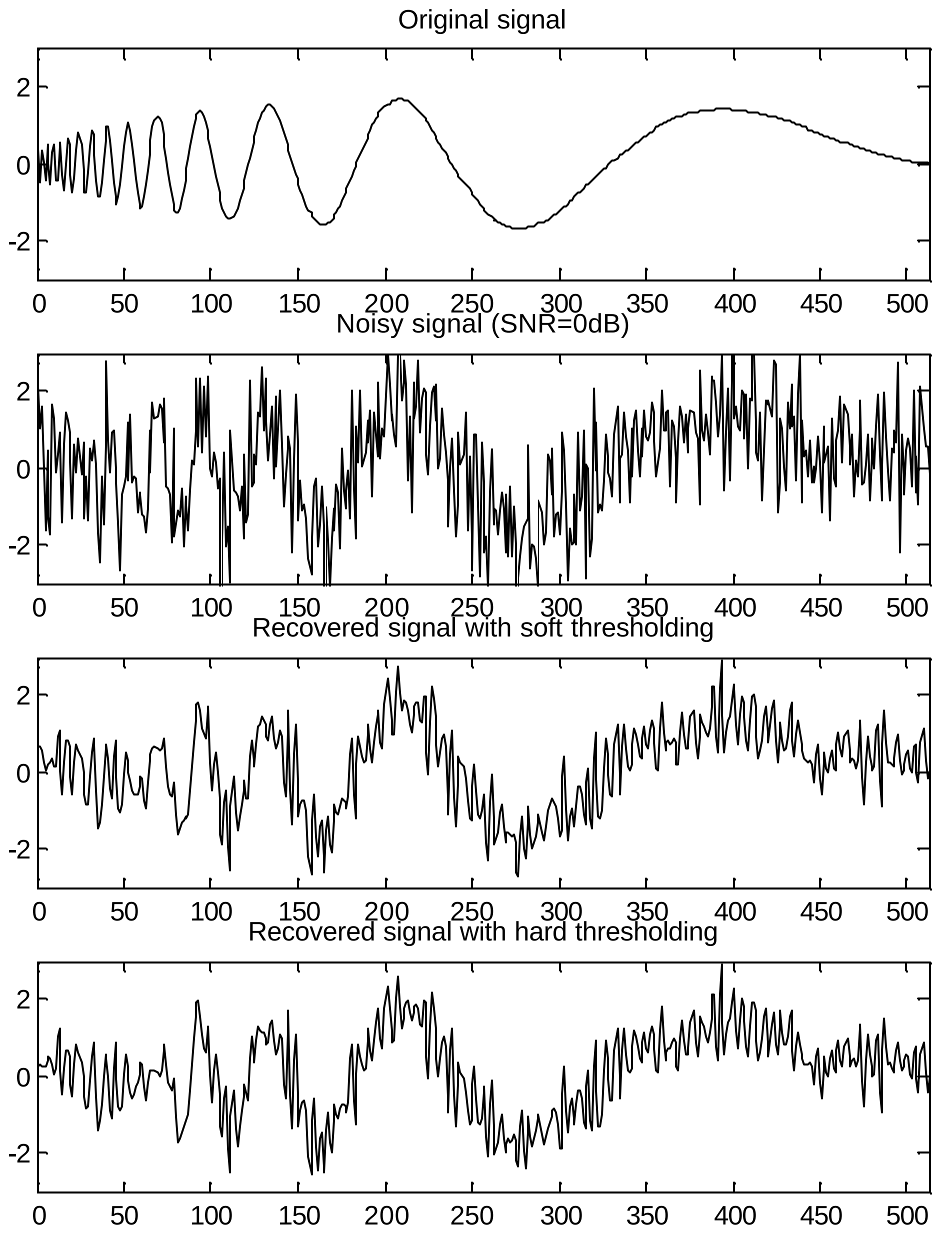

Figure 35. WT-based denoising for increasing amplitude chirp type, 4 level, method 1, Daubechies wavelet 5, SNR=0 dB.

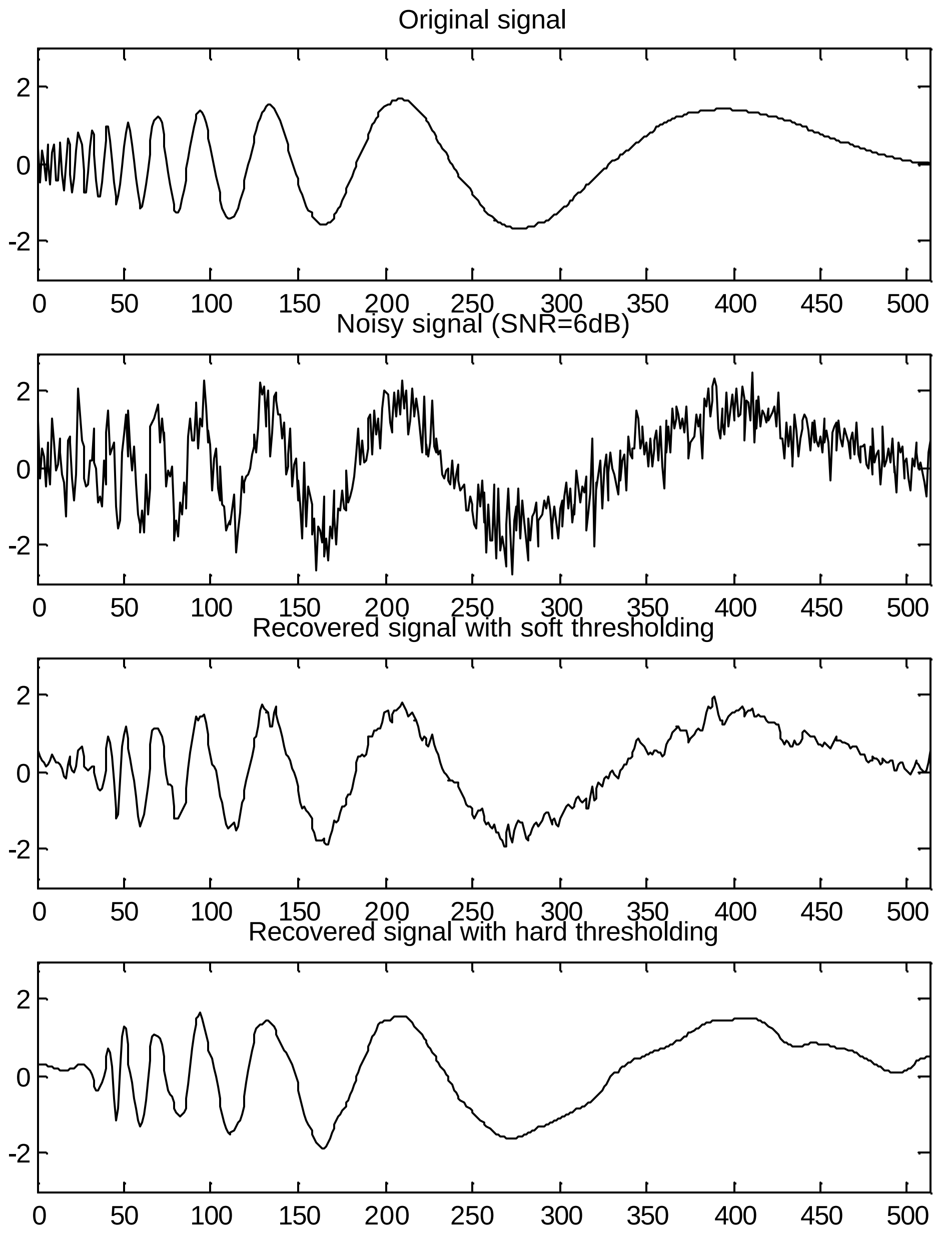

Figure 36. WT-based denoising for increasing amplitude chirp type, 4 level, method 1, Daubechies wavelet 5, SNR=6 dB.

APPENDIX C. COMPARISON WITH WAVELET SHRINKAGE ALGORITHM

The proposed denoising schemes are compared with the original Wavelet shrinkage algorithm in this chapter. Wavelet shrinkage algorithm is a well-known denoising scheme.

A. WAVELET SHRINKAGE ALGORITHM

The Wavelet shrinkage algorithm was introduced by Donoho and Johnstone [16, 17] to denoise signals embedded in additive white Gaussian noise with unit variance. This algorithm has three steps:

a. Wavelet transformation.

b. Noisy coefficients suppression by applying a non-linear thresholding technique.

c. Inverse wavelet transformation.

1. Choosing a Threshold Value

The following four threshold selection schemes are available:

a. Stein's Unbiased Risk Estimator (SURE) Threshold

The SURE threshold value is derived adaptively for each decomposition level by minimizing the Stein's Unbiased Estimate of risk [18] for threshold estimates.

b. Sqtwolog Threshold

This method uses a fixed threshold value defined as:

$$T = \sqrt{2\log(length(signal))}\ .$$

c. Heursure Threshold

This method is a mixture of the preceding two methods. It uses the Sqtwolog threshold at low SNR levels and the SURE threshold at medium to high SNR levels.

d. Minimax Threshold

This method uses a fixed threshold that gives minimax performance for the MSE. The minimax principle is used in statistics in order to design estimators, which realize the minimum of the maximum mean square error obtained for the worst function in a given set. So, minimax threshold is the value that provides the minimum MSE among the worst threshold values for the detail and approximation coefficient sets.

2. Thresholding Methods

a. Hard Thresholding

This method sets the coefficients with absolute values below the chosen threshold to zero.

b. Soft Thresholding

This method first sets the coefficients with absolute values below a chosen threshold to zero. Then, it shrinks the remaining coefficients using the relationship $\hat{c} = \mathrm{sign}(c)[|c| - T],$ where c is the coefficient to be thresholded, $\hat{c}$ is the thresholded coefficient, and T is the chosen threshold value.

B. COMPARISONS

Simulations were performed, using the wavelet shrinkage algorithm defined above, on the same signals that were used for the proposed scheme. However, different SNR values were obtained by using constant noise power and different values for the signal power in this experiment, contrary to what was done in the simulations considered for the proposed scheme in the main body of the thesis. The Heursure method was used to compute the threshold values for the wavelet shrinkage algorithm. A new simulation was performed with the proposed scheme under the same SNR conditions as well. Results shown in Figures 37 to 39 indicate that the proposed WT-based denoising scheme performs better than the wavelet shrinkage algorithm for the sinusoidal test signal. However, the wavelet shrinkage algorithm outperforms the proposed scheme for the chirp type test signals. It should be noted that these results are restricted to the case of noise with unit variance.

1. Sinusoidal Signal

Results shown in Figure 37 indicate that the MSE performances of the two denoising algorithms are similar, whereas the distance measure *d* shows that the proposed denoising scheme outperforms the wavelet shrinkage algorithm at low SNRs.

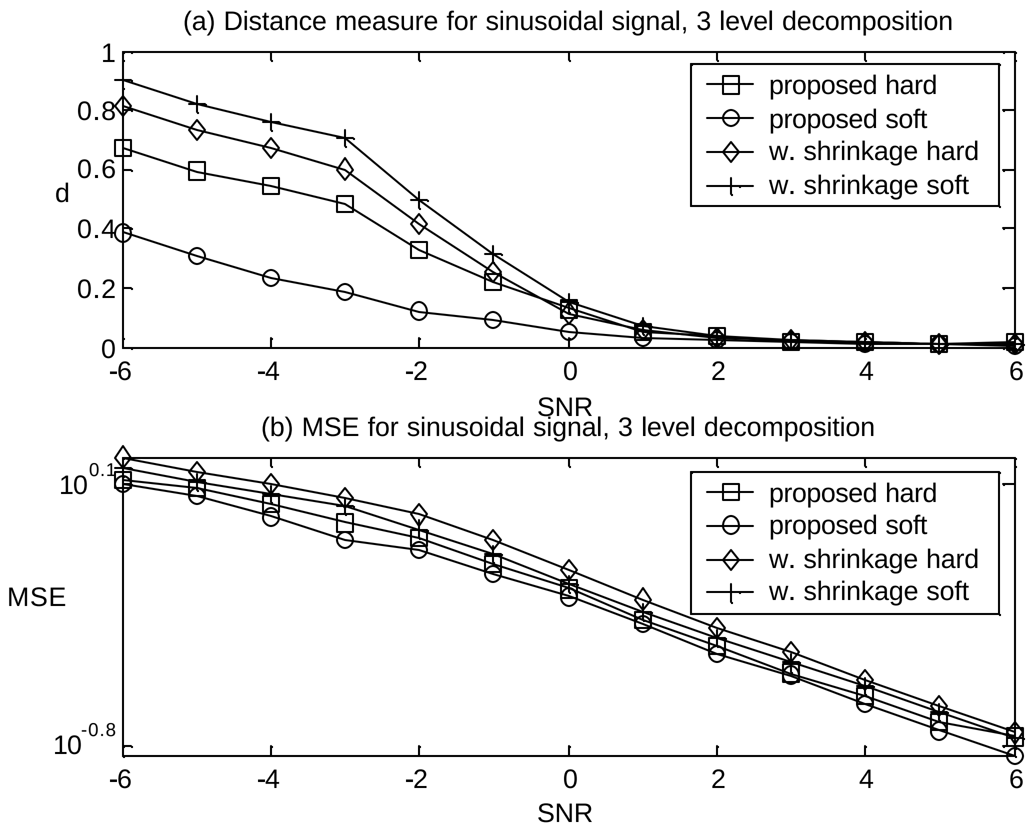

Figure 37. Wavelet shrinkage algorithm and proposed WT-based denoising scheme performances for the sinusoidal signal type, using 3-level decomposition and Daubechies wavelet 5.

2. Constant Amplitude Chirp

Both performance criteria show that the wavelet shrinkage algorithm outperforms the proposed denoising scheme for this signal. This can be seen in Figure 38.

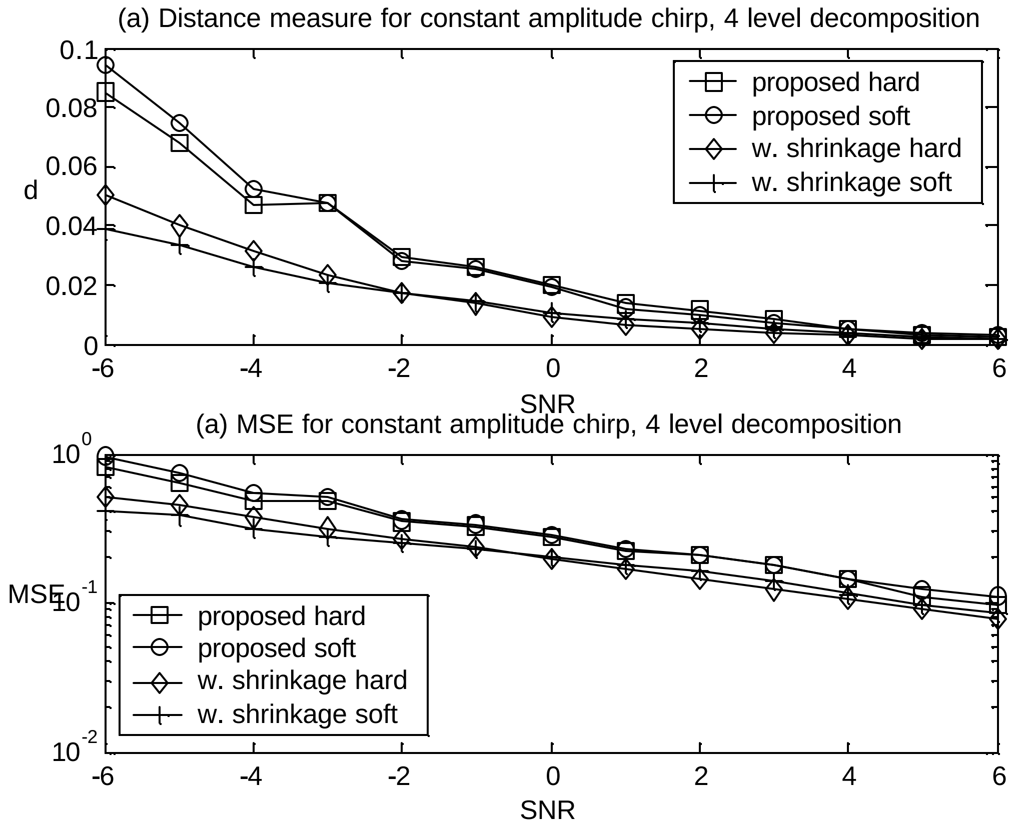

Figure 38. Wavelet shrinkage algorithm and proposed WT-based denoising scheme performances for the constant amplitude chirp type, using 4-level decomposition and Daubechies wavelet 4.

3. Chirp With an RC Time Constant-Like Amplitude Increase

Both performance criteria show that the wavelet shrinkage algorithm outperforms the proposed denoising scheme for this signal as shown in Figure 39.

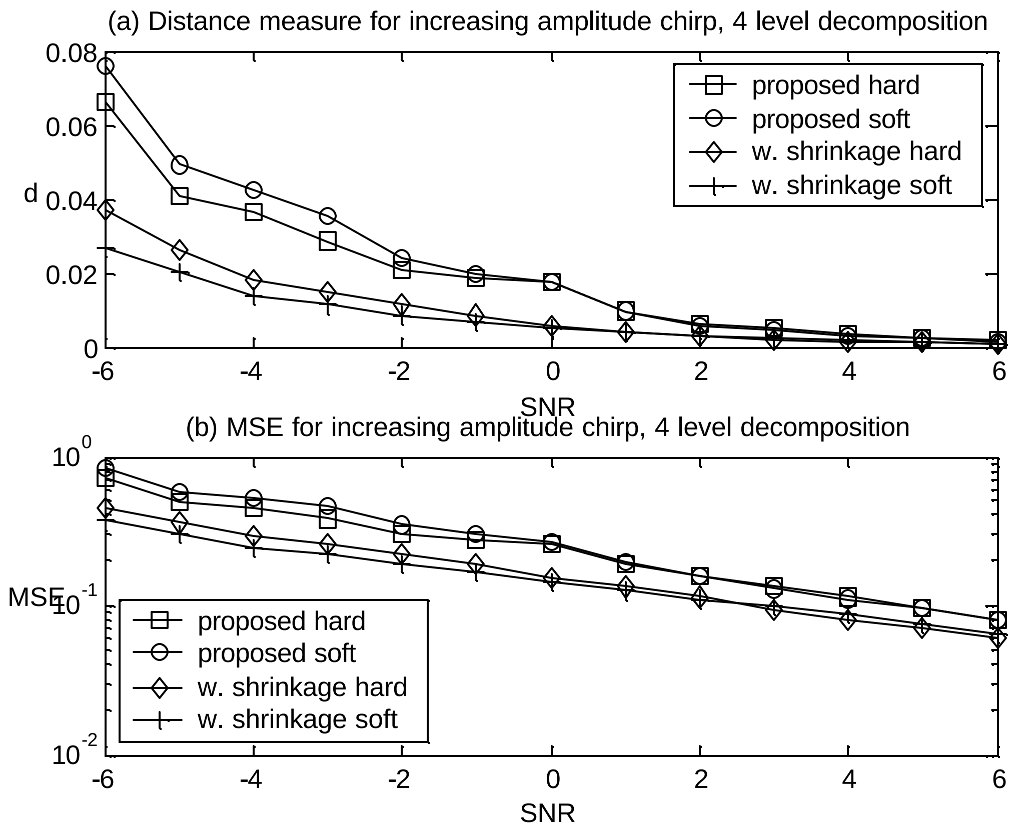

Figure 39. Wavelet shrinkage algorithm and proposed WT-based denoising scheme performances for the increasing amplitude chirp type, using 4-level decomposition and Daubechies wavelet 5.

It should be noted that the new proposed method (based on kurtosis and Bootstrap method) seem to be inferior to the methods described in this appendix. But the methods in this appendix need additional information: the variance of the noise is unity, which may be an unreasonable constraint in some practical applications.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Oppenheim, A.V. and Willsky, A.S., *Signals and Systems*, 2nd ed., Prentice Hall, 1997.
2. Proakis, J.G. and Manolakis, D.G., *Digital Signal Processing*, 3rd ed., Prentice Hall, 1996.
3. Hlawatsch, F. and Boudreaux-Bartels, G.F., "Linear and Quadratic Time-Frequency Signal Representations," *IEEE Signal Processing Magazine*, Vol. 9, No. 2, pp. 21-67, April 1992.
4. Burrus, C.S., Gopinath, R.A., and Guo, H., *Introduction to Wavelets and Wavelet Transforms, A Primer,* Prentice Hall, 1998.
5. Rioul, O. and Vetterli, M., "Wavelets and Signal Processing," *IEEE Signal Processing Magazine*, Vol. 8, No. 4, pp. 14-38, October 1991.
6. Hippenstiel, R.D., *Detection Theory, Applications and Digital Signal Processing,* CRC Press, 2002.
7. Johnson, R.A., *Probability and Statistics for Engineers,* 6th ed., Prentice Hall, 2000.
8. Zoubir, A.M. and Boashash, B., "The Bootstrap and its Application in Signal Processing," *IEEE Signal Processing Magazine,* Vol. 15, No. 1, pp. 56-76, January 1998.
9. Hall, P. and Wilson, S.R., "Two Guidelines for Bootstrap Hypothesis Testing," *Biometrics*, Vol. 47, pp. 757-762, June 1991.
10. Aktas. U., *Time Difference of Arrival (TDOA) Estimation Using Wavelet Based Denoising*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1999.
11. Mantis, S.D., *Localization of Wireless Communication Emitters Using Time Difference of Arrival (TDOA) Methods in Noisy Channels*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 2001.
12. Ravier, P. and Amblard, P., "Denoising Using Wavelet Packets and the Kurtosis: Application to Transient Detection," *Proceedings of the 1998 IEEE-SP International Symposium on Time-Frequency and Time-Scale Analysis*, pp. 625-628, 1998.
13. Donoho. D., Duncan M.R. and Huo X., "Wavelab 802," October 1999, [http://www-stat.stanford.edu/~wavelab/], last accessed September 2002.

14. The MATLAB Software version 6.1, The Mathworks Inc., May 2001.
15. Zoubir, A.M. and Iskander D.R., "Bootstrap Matlab Toolbox Version 2.0," May 1998, [http://www.atri.curtin.edu.au/csp/downloads/bootstrap_toolbox.html], last accessed August 2002.
16. Donoho, D. and Johnstone I., "Ideal Spatial Adaptation via Wavelet Shrinkage," *Biometrika*, Vol. 81, pp. 425-455, 1994.
17. Donoho, D., "De-Noising by Soft-Thresholding," *IEEE Transactions on Information Theory*, Vol. 41, No. 3, pp. 613-627, May 1995.
18. Stein, C., "Estimation of the Mean of a Multivariate Normal Distribution," *The Annals of Statistics*, Vol. 9, pp. 1135-1151, 1981.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Chairman, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

4. Prof. Monique P. Fargues
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

5. Prof. Ralph D. Hippenstiel
   Chairman
   Department of Electrical Engineering
   University of Texas at Tyler
   Tyler, Texas

6. Prof. Roberto Cristi
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

7. Kara Harp Okulu
   Bakanliklar, Ankara, TURKEY